The Burrow of the Marmots

Vlad Coroian, Anda Lăzărescu,
Alexandra Croitoriu, Alexandra Rus, Nora Petruța
Colegiul Național "Emil Racoviță" (Cluj, România)
Paul Comte, Thierno Bah
Lycée d'Altitude (Briançon, France)
Teacher: Ariana Văcărețu, Thierry Jayle
Researcher:
Lorand Parajdi, Universitatea "Babeș Bolyai"

2018-2019

1 Subject description

This project describes a solution to one of the MATh.en.JEANS research topic and an algorithm designed to solve the generalization of the problem.

A group of marmots decides to build a new burrow, in sight of the upcoming winter, but this year they've decided to make it in an optimized way. The problem of these marmots is that their sleep is very light, this implicating a set of 3 rules so that the structure doesn't collapse:

- Starting from the entrance or from the end of a corridor, there can be built only a maximum of 2 corridors, otherwise the burrow collapses.
- It is inconceivable for a marmot to sleep at a crossroad or in the middle of one corridor. If it did that, other marmots which live further in the burrow would have to go over it and this would ruin their hibernation. So marmots only sleep at the end of a corridor which only leads to an only room.
- Even the simple movement and the noise of their tiny paws generates vibrations which disturb the group during their sleep (they really do have a light sleep!!!), and knowing how many times each of the marmots will wake up and get out of the burrow during the winter, we will make sure that the sum of their movements is as little as possible.

For example, a marmot which wakes up 6 times and is 4 corridors away from the exit will have to cover $6 \cdot 4 = 24$ corridors, both ways (but for working with lower numbers we will only consider the "going"). If we lay the marmot at 1 corridor away from the exit she won't have to cover more than 6x1=6 corridors.

How can we build a burrow for the following marmot family: M_1(6 awakenings), M_2(4), M_3(4), M_4(1), M_5(3)?

2 Results

This research paper contains the solution for a generalization of the problem stated. It also provides an algorithm designed to find the solution for a given set of marmots.

3 Extension for the general case

The problem presented can be extended to a general case, by trying to construct the burrow for a given set of n marmots, with their respective number of awakenings $M_1, M_2, .., M_n$. The algorithm designed for this generalization can therefore be applied to the particular case originally stated in the problem.

4 Defining the notions used

4.1 Notations

We will consider that at the end of each corridor of the burrow there is a room. The exit is also considered a room. Therefore, there are three types of rooms:

- Intersections (used to separate a corridor in two other corridors, further away from the exit)
- Room of a marmot (placed at the end of a sequence of corridors)
- Exit from the burrow

A full binary tree is a binary tree in which every node other that the leaves has two children. Hence, we can also consider the burrow of the marmots a full binary tree, where the leaves are the marmots, which have no sons and the inner nodes are the exit and the intersections, which all have exactly two sons.

A subtree of a tree T is a tree consisting of a node in T and all of its descendants in T. In our problem, in particular, the subtree of a node N consists of the nodes representing rooms of marmots which have to go through N to get to the exit and all the inner-nodes (intersections) between the marmots' rooms and N.

We now have n marmots, so the full binary tree has exactly n leaf nodes. In a Full Binary Tree, the number of leaf nodes is the number of internal nodes plus 1. Therefore, the full binary tree in the problem stated has exactly $2n-1$ nodes, n leaf nodes (marmots) and $n-1$ inner nodes (intersections and exit node).

We will define the total value of a tree T as the sum of the movements of the marmots in the tree and we will use the notation V_T.

We define $Level_i =$ the level of the node which represents the $i-th$ marmot's room.

We define the cost of a node as following:

(a) $Cost_{leaf} =$ the number of awakenings corresponding to the marmot

(b) $Cost_{inner-node} =$ the sum of awakenings corresponding to the marmots in the subtree of that inner-node (Obs: This is also equal to the sum of the costs of the two children of the inner-node)

We also define the weight of a marmot as

$$Weight_i = Cost_{leaf_i} \cdot Level_i, \forall i = \overline{1, n}$$

4.2 Calculating the total value for a tree

This weight represents the distance the marmot has to travel to get to the exit times the number of awakenings it has. Therefore, the total value for a tree is the sum of all the weights of the marmots. So this is the sum we try to minimize in our problem.

We will also present an easier way of finding the total value of a tree. This result can also be written as the sum of the costs of all the inner-nodes present in our binary tree. This can be proven using induction.

4.3 Proof

$P(n)$: The value of a tree with n leaves is equal to the sum of the costs of all intersections.

There are two cases for our induction, because the tree is either a leaf or it consists of two full binary subtrees and a root.

For the first case, we only have one marmot. This marmot can be placed right at the exit of the burrow, so the total value is 0 (In this case we have no inner-nodes). Then $P(1)$ is true. Moreover, if we have two marmots, the only suitable structure for the burrow is the following:

Because each marmot has one corridor to exit and they wake up M_1 and M_2 times, the total value for this tree is $M_1 + M_2$. Then $P(2)$ is also true, because there is only one inner-node with cost $M_1 + M_2$.

We can now assume that $P(1), .., P(n)$ are true and we prove $P(n+1)$. Let's assume we have $n+1$ marmots with the number of awakenings $M_1, M_2, .., M_{n+1}$.

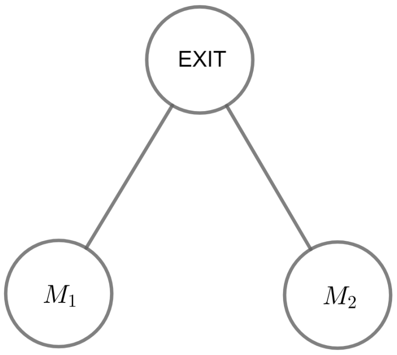

Figure 1: Full Binary Tree with two leaves

Let T be an arbitrary full binary tree with $n+1$ leaves. T consists of a root R and two subtrees, T_1, T_2, with roots R_1 and R_2 (R_1 and R_2 are the children of R). We use the following notations:

- n_1 and n_2 the number of leaves $n_1, n_2 \geq 1$, $n_1 + n_2 = n + 1$
- C_1 and C_2 the costs in R_1, R_2
- V_{T_1} and V_{T_2} the total values for the two subtrees

Because $P(n_1)$ and $P(n_2)$ are both true, each marmot from the two subtrees has to walk one more corridor each time it wakes up. $C_1 + C_2$ represents the sum of awakenings from all the marmots in subtrees T_1 and T_2, so the total value for tree T is equal to $V_{T_1} + V_{T_2} + (C_1 + C_2)$.

Root R has the cost $C_R = C_1 + C_2$, so the total value should be $V_T = V_{T_1} + V_{T_2} + C_R$.

Therefore, we can conclude that $P(n+1)$ is also true and the induction is over.

We now have that $P(n)$ is true $\forall n \in \mathbb{N}^*$ and we have found an easier way to calculate the total value for a tree.

5 Solution for the generalization

We now want to find a way to construct a tree with minimal total value. We also observe that if we place a marmot further from the exit, it influences the final answer more. Therefore, we try to place the marmots with the smallest costs first.

We make the following observation: If we have grouped two marmots together as in Figure 2, we can consider it a new "marmot" and insert it in the current set of marmots with value $M_1 + M_2$ (the structure of this "marmot" does not change, however). This is true because of the property we proved at 4.3. If at the beginning we had n marmots, we now only have $n - 1$ marmots. So if we continue this algorithm for n steps we can construct a tree, which represents the burrow of the marmots.

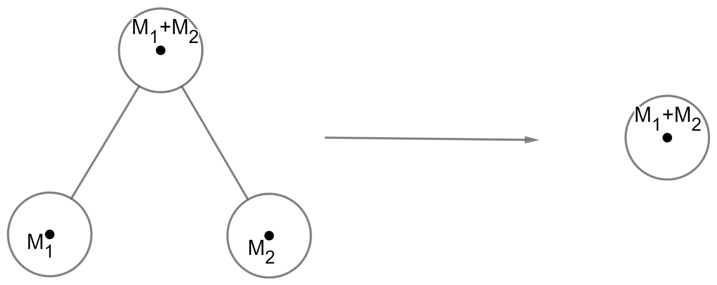

Figure 2: Grouping two marmots

We have to decide now how to group the marmots together. We always choose two marmots (even though the "marmots" we choose may come from previous groupings). Moreover, we always try to use the marmots with the smallest values for a group (because when we group them, we actually place them further from the root). [1]

We can now describe the algorithm we can use to find the smallest sum of movements:

```
initialization;
answer=0;
while there are at least 2 marmots do
    find the marmots with the littlest number of awakenings;
    $a$=smallest value in set;
    $b$=second smallest value in set;
    remove marmots a and b from the set of marmots;
    insert a new marmot in the set with value $a + b$;
    $answer = answer + (a + b)$;
end
```

Algorithm 1: Finding the smallest sum of movements

After executing this algorithm, the minimal sum of movements is the value of variable answer. To construct the burrow we can trace back the groups we have made.

6 Approach for the particular case

We will apply the algorithm described for the case given in the statement of the problem. Here we have $n = 5, M_1 = 6, M_2 = 4, M_3 = 4, M_4 = 3, M_5 = 1$.

1. The first step is to select the two smallest values from the original values, 6, 4, 4, 3, 1. These are 3 and 1. We remove these from the set and insert a new "marmot" whose value is 3+1=4.
2. We now have 4 marmots left and the values are 6, 4, 4, 4. Because now we have more choices for a group of marmots, we will have more possible structures which have the minimal total value. We now choose marmots with values 4 and 4. We replace them with a new marmot with value 4+4=8.
3. Now we only have three marmots left, 8, 6, 4. We choose 6 and 4, remove them from the set and insert a new value equal to 6+4=10.
4. The only marmots left are 10 and 8. We replace them with their sum, 10+8=18. Here our algorithm stops.

The minimal sum of movements is the sum of the marmots inserted at each step. This is equal to 4+8+10+18=40. We can obtain this sum, by constructing the trees in Figure 3 and Figure 4. Of course, there are multiple different trees which we can build, but those are obtained by permutations of some subtrees of the trees in Figure 3 and Figure 4.

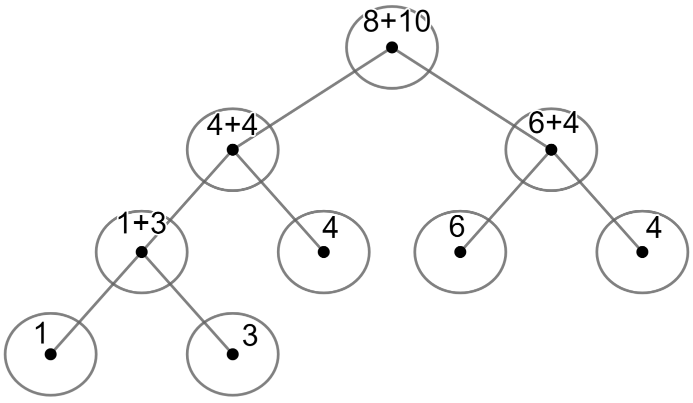

Figure 3: Structure 1

7 Implementation details for the algorithm

Here is a C++ implementation of the algorithm designed to find the minimal total value of a tree. To implement it, we used a multiset, which we used to store the marmots' value, including the new marmots inserted and possible duplicates. This algorithm complexity is $O(N \cdot log_2 N)$, where N is the number of marmots. [2] [3]

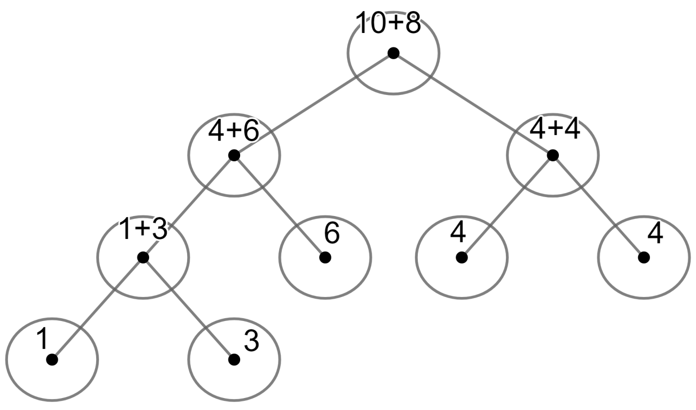

Figure 4: Structure 2

```
#include <fstream>
#include <set>

using namespace std;
ifstream fi("marmots.in");
ofstream fo("marmots.out");
const int NMAX=1e6+5;
int n,M[NMAX],answer;
multiset <int> marmots;
int main()
{
        fi>>n;
        for(int i=1;i<=n;i++)
        {
                fi>>M[i];
                marmots.insert(M[i]);
        }
        while(n-->1)
        {
                int a=(*marmots.begin());
                marmots.erase(marmots.lower_bound(a));
                int b=(*marmots.begin());
                marmots.erase(marmots.lower_bound(b));
                int new_marmot=a+b;
                marmots.insert(new_marmot);
                answer+=new_marmot;
        }
        fo<<"The minimal value of movements is : "<<answer;
        fi.close();
        fo.close();
        return 0;
}
```

The following algorithm describes a possible structure which gives the minimal sum. In the output *−i−* represents an intersection, *−root−* represents the exit from the burrow and $-(x)-$ represents a marmot with value x. To construct the burrow, we have defined a recursive function *structure*, which calls itself until every marmots is printed in the output file.

```
#include <fstream>
#include <set>
using namespace std;
ifstream fi("marmots.in");
ofstream fo("marmots.out");
const int NMAX=1e6+5,INF=1e9;
pair <int,int> in[NMAX];
int n,M[NMAX],answer;
multiset <pair<int,int>> marmots;
void structure(int nr)
{
        if(in[-nr].first<0)
        {
                fo<<"{";
                structure(in[-nr].first);
                fo<<"}";
        }
        else fo<<"("<<M[in[-nr].first]<<")";

        if(nr==-n+1) fo<<"-root-";
        else fo<<"-i-";

        if(in[-nr].second<0)
        {
                fo<<"{";
                structure(in[-nr].second);
                fo<<"}";
        }
        else fo<<"("<<M[in[-nr].second]<<")";
}
```

```
int main()
{
        fi>>n;
        for(int i=1;i<=n;i++)
        {
                fi>>M[i];
                marmots.insert({M[i],i});
        }
        for(int i=1;i<n;i++)
        {
                pair <int,int> a=(*marmots.begin());
                marmots.erase(marmots.lower_bound(a));
                pair<int,int> b=(*marmots.begin());
                marmots.erase(marmots.lower_bound(b));
                int new_marmot=a.first+b.first;
                in[i]={a.second,b.second};
                int index=-i;
                marmots.insert({new_marmot,index});
                answer+=new_marmot;
        }
        fo<<"The minimal value of movements is: "<<answer<<"\n";
        structure(-(n-1));
        fi.close();
        fo.close();
        return 0;
}
```

8 Conclusions and observations

There are multiple interesting observations for the way the burrow is constructed. Firstly, we can prove that the number of possible burrows with n marmots is C_n, the $n-th$ Catalan number. [4] Therefore, we can also give a recurrence relation between the number of burrows.

$$C_n = \binom{2n}{n} - \binom{2n}{n+1}$$

$$C_{n+1} = \sum_{i=1}^{n} C_i C_{n-i}$$

We could also try to solve another kind of generalization. In this case, we could change the rules to construct a burrow. The burrow could take the form of a $k - ary$ tree, so each corridor could be divided into exactly k other corridors. A similar strategy could be applied, where instead of choosing the smallest two values, we choose the smallest k values and replace them with a new "marmot" which has the number of awakenings equal to the sum of the marmots chosen. We should also take into account the case where the number of marmots is not divisible by k and make sure we place the marmots as close as possible to the exit.

9 Bibliography

1. https://www.geeksforgeeks.org/binary-tree-set-3-types-of-binary-tree/
2. https://en.wikipedia.org/wiki/Binary_tree

10 Editing notes

[1] The fact that the algorithm is correct must be proved. Here is a proof:
Lemma: There exists an optimal tree where two marmots with lowest awakenings are leafs of the same inner node.

The main idea is an exchange argument: if a marmot with more awakenings is put at a higher level than an another one with less awakenings then, by exchanging these two marmots one decreases the total weight. This proves that, in an optimal tree, there is a marmot with a minimal cost in the highest level. Then, exchanging eventually its neighbour with one of minimal cost among the others, we do not increase the total weight, so there exists an optimal tree where two marmots with lowest awakenings are leafs of the same inner node.

Proposition: The tree H built by the algorithm has a total value $\Sigma(H)$ minimal.

The proof is by induction on the number n of marmots. Roughly speaking: if this is true for $n-1$ marmots, we can apply this induction hypothesis after merging two marmots with minimal cost and then see that after separating these two marmots again the tree remains optimal. For more details: the case $n = 1$ is trivial. Assume the property is true for any set of $n-1$ marmots.
Let $x_1 \leq x_2 \leq ... \leq x_n$ be the awakenings of a set of n marmots, H the tree built by the algorithm and T a minimal tree. By construction x_1 and x_2 are leafs of a same inner-node in H. By lemma we may assume that x_1 and x_2 are also leafs of a same inner-node in T. Now remove the leafs containing x_1 and x_2, making their father a new leaf (with value $x_1 + x_2$) in H and T. We obtain trees H' and T' on the set $\{x_1 + x_2,\ x_3,\ ...\ , x_n\}$. H' is the tree built by the algorithm. $\Sigma(H') = \Sigma(H) - x_1 - x_2$ and $\Sigma(T') = \Sigma(T) - x_1 - x_2$. By the induction hypothesis ($\Sigma(H')$ is minimal) $\Sigma(H') \leq \Sigma(T')$ and then $\Sigma(H) \leq \Sigma(T)$. So $\Sigma(H) = \Sigma(T)$ and $\Sigma(H)$ is minimal.

[2] In multiset C++ STL, search, removal, and insertion operations have logarithmic complexity because multisets are implemented as red-black (or AVL) binary search trees. Then the complexity of for (int i=1; i<= n; i++) line 13 is $\Theta(n\ log_2\ n)$. The complexity of removing 2 values and inserting a single value (lines 21 to 25) is $\Theta(log_2\ n)$. So the complexity of while (n-- <1) line 18 is also $\Theta(n\ log_2\ n)$. Then the complexity of the algorithm is effectively $\Theta(n\ log_2\ n)$.

[3] About the line 9 of the next program we can notice that Min Heap are more suitable structures than multiset (https://fr.wikipedia.org/wiki/Tas_binaire).
[4] The number of burrows is equal to C_{n-1}, not to C_n. The interested reader can consult the reference https://fr.wikipedia.org/wiki/Arbre_(combinatoire) for a result without proof. On the MATh.en.JEANS website one can find also papers on this topic, namely
https://www.mathenjeans.fr/sites/default/files/comptes-rendus/article_arbres_michelet_jolimont.pdf
https://www.mathenjeans.fr/sites/default/files/comptes-rendus/article_triangulation-michelet-jolimont_2017.pdf)