THE BURROW OF THE MARMOTS

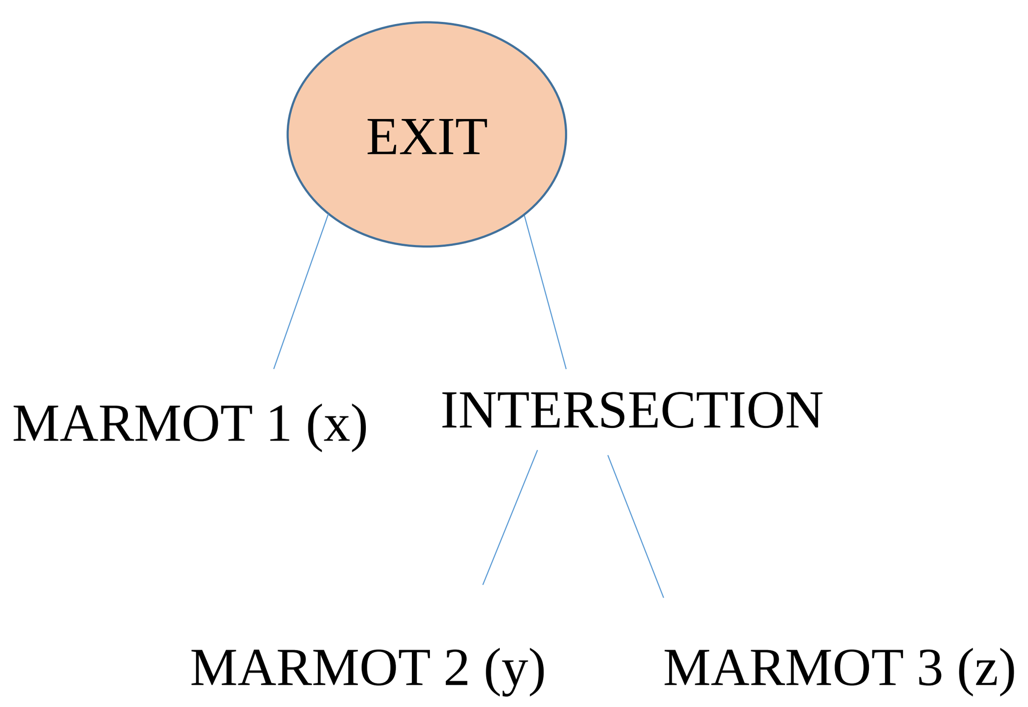
Vlad COROIAN, Anda LĂZĂRESCU,
Alexandra CROITORIU, Nora PETRUȚA, Alexandra RUS
Colegiul Național “Emil Racoviță” Cluj-Napoca, Romania

The research topic

A group of marmots decides to dig a new burrow for the winter that is coming, but this year they decided to make it in a more optimized way. Their problem is that they have a soft sleep (they are light sleepers) that involves 2 rules and another one for the construction not to fall down:

- Starting from the entrance or the extremity of the corridor, we can build no more than 2 corridors, or else the construction risks to fall down.
- It is inconceivable for a marmot to sleep at a crossroad or in the middle of the corridor. If so, they should walk above the others who live at a lower level, which means ruining their hibernation. In this way, the marmots sleep in the bottom of the corridor, that also represents the only way out of the construction.
- Even the basic movement of the marmots and the sound of their steps generate vibrations that disturbs the group from their sleep (they really sleep soft). So, if we know how many times each marmot wakes up, we will make sure that the sum of the movements is minimum. For example, a marmot that wakes up for 6 times and there are 4 corridors it will be 6x4=24 corridors, but in order to have smaller numbers, we will count only the way forward. If the marmot is in the exit corridor, it will make only 6x1=6 corridors. How do we build the burrow of the marmots for the next families: M_1(6 wakes), M_2(4), M_3(4), M_4(1), M_5(3)?

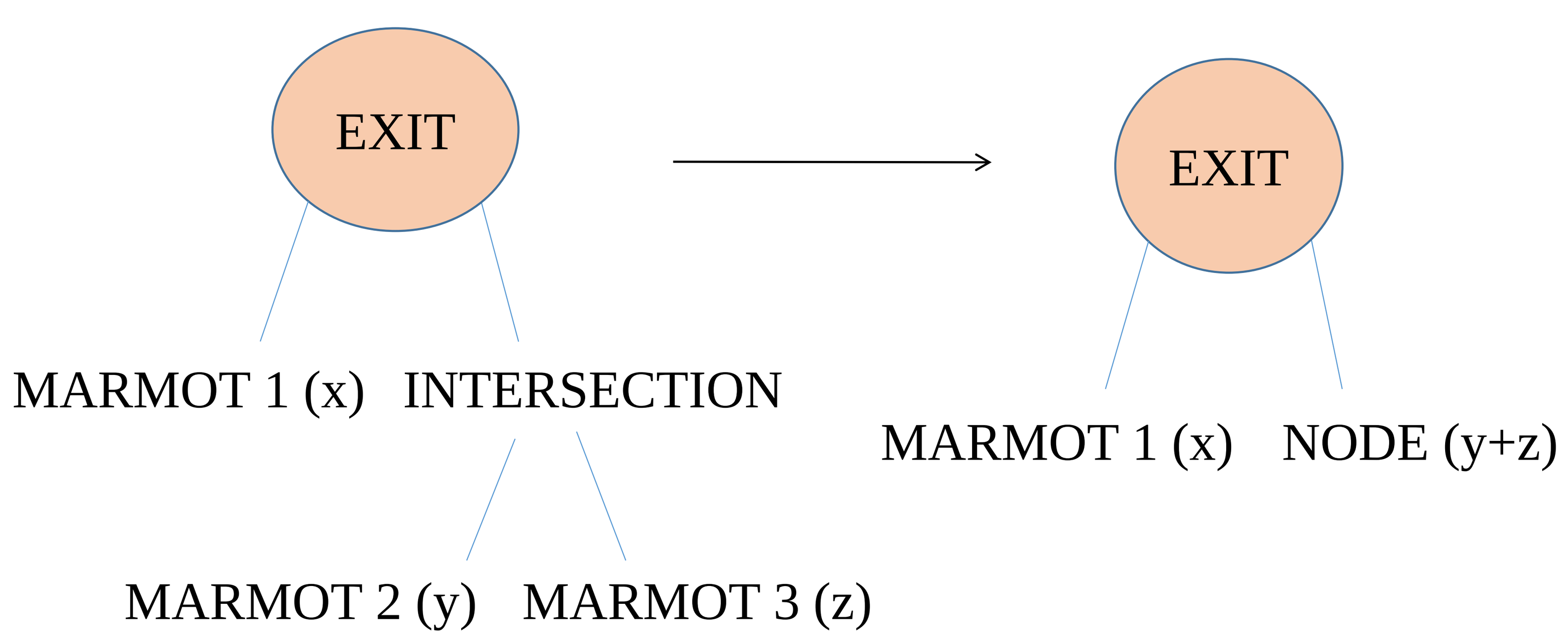
Representations



- Full bianry tree (each node is either an intersection, which has 2 sons, or a leaf, a marmot);
- Each edge represents a corridor;
- There are **n** leaves, **2n-1** nodes, **n-1** intersections;
- Every leaf is asociated a cost (which is the number of awakenings);
- Weight of a node= (number of awakenings for a marmot) * (distance from the exit).

Demonstration for the general case

- We associate each internal node (intersection) the sum of the values in its **2** sons;



- The cost of the whole tree is now the sum of the values in these intersections. We can prove this by induction.

Proof

- **n=1**, the cost is 0;
- **n=2**, one intersection with the value **V1+V2**; the final cost is **V1+V2**;

  ⇒ the statement is true for n=1 and n=2;
- We can now assume that the statement is true for **k=1, k=2, …, k=n** and we prove it for **n+1**.

We consider an arbitrary full binary tree with 2 sons:

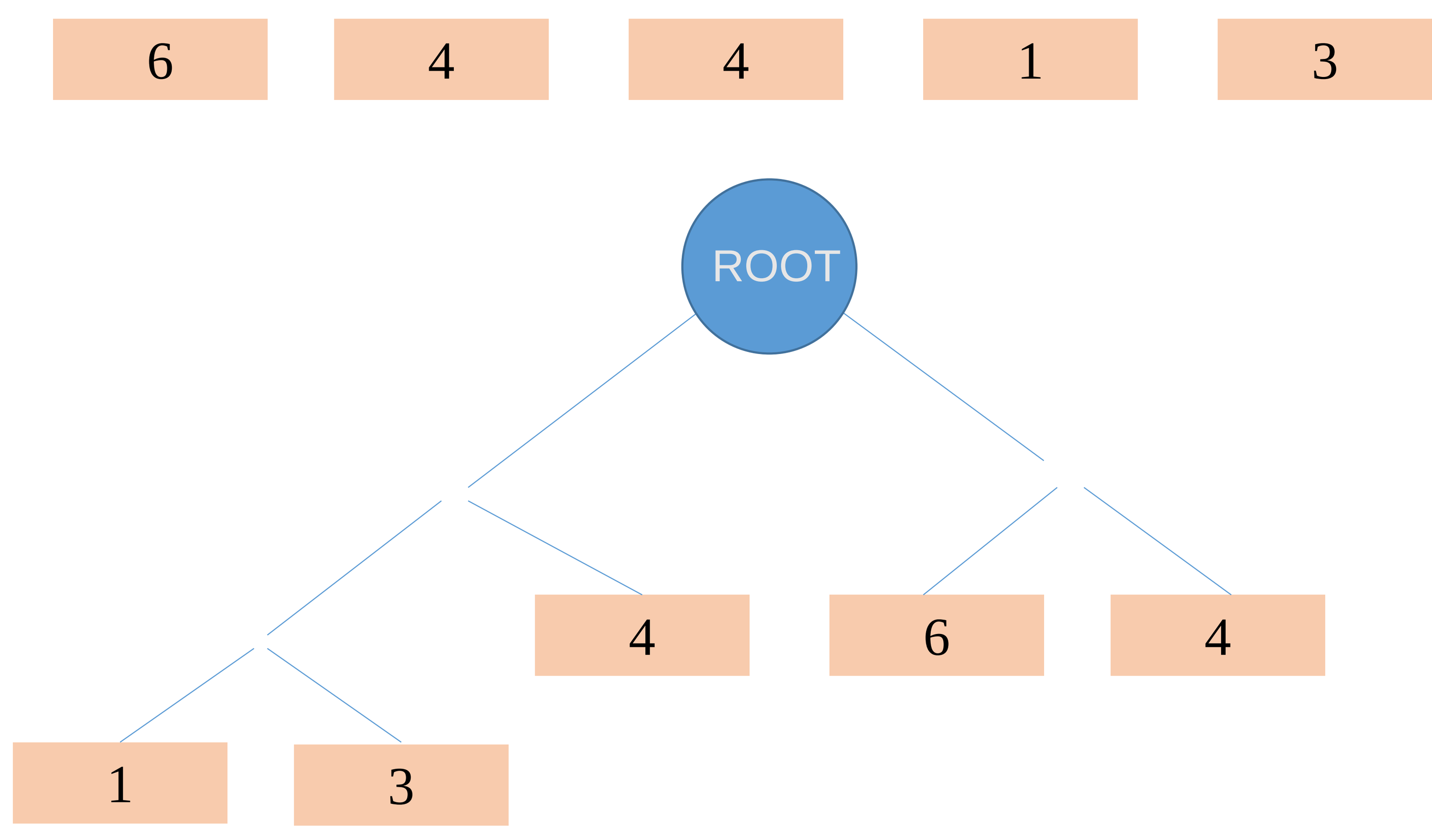
Left one: – **k** marmots;
– **S1**=total sum of awakenings of the left subtree;
– **C1**=total cost of the left subtree;

Right one: – **n+1-k** marmots;
– **S2**=total sum of awakenings of the left subtree;
– **C2**=total cost of the right subtree;

⇒ The value in the root will be **S1+S2** and the total cost for the tree will be **(C1+C2)+(S1+S2)**.

⇒ So the statement is true for **n+1** too.

Solution of the general case



The solution is 1*3+3*3+4*2+6*2+4*2=40

The algorithm

- Greedy strategy – obtaining the minimal sum at each step;
- Using marmots with smaller values first;
- **n** marmots – **n** leaf nodes;
- Grouping together the 2 smallest values;
- Repeating the process until only one marmot is left, whose value will actually be the answer to our problem;

1. We use a set to memorize the values we currently have and another variable to store the answer.

   Complexity: O(N^2*logN);
2. We use 2 deques for the marmots we have initially and for the “new marmots” we insert.

   Complexity: O(N*logN).

We can also provide an algorithm for generating the structure, by storing additional data as we do the steps mentioned before.

```cpp
void structure(int nr)
{
    if(in[-nr].first<0)
        fo<<"{",structure(in[-nr].first),fo<<"}";
    else fo<<"("<<s[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<<"("<<s[in[-nr].second]<<")";
}
int main()
{
    fi>>n;
    for(int i=1;i<=n;i++)
    {
        fi>>m[i].first;///m[i].first=number of awakenings for marmot i
        m[i].second=i;///m[i].second=the indice before sorting of the marmot i
        s[i]=m[i].first;
    }
    sort(m+1,m+n+1);
    for(int i=1;i<n;i++)
    {
        m[1].first=m[1].first+m[2].first; /// we give m[1] the value of the new intersection created
        in[i]={m[1].second,m[2].second};
        m[1].second=-i; m[2].first=INF;
        sort(m+1,m+n+1);
    }
    structure(-(n-1));
}
```

```cpp
int next()
{
    int rez;
    if(Q.empty())
    {   rez=R[0]; R.pop_front(); return rez;}
    if(R.empty())
    {   rez=Q[0]; Q.pop_front(); return rez;}
    if(R[0]<Q[0])
    {   rez=R[0]; R.pop_front(); return rez;}
    rez=Q[0]; Q.pop_front(); return rez;
}
int main()
{
    ///n=number of marmots
    fi>>n;
    for(int i=1;i<=n;i++)
        fi>>m[i]; ///m[i]=number of awakenings for marmot i
    sort(m+1,m+n+1);
    for(int i=1;i<=n;i++)
        Q.push_back(m[i]);
    ///in deque Q we have the values of the actual marmots and not the values of the intersections (sorted)
    while(R.size()+Q.size()>1)
    {
        int a=next(),b=next();
        ///a and b are the smallest values we currently have
        s+=a+b;
        R.push_back(a+b);///grouping the marmots with values a and b
        ///in deque R we have the values for all the intersections we use
    }
    fo<<s;///s=answer to our problem, the sum of weights
    ///In this algorithm we find the minimal sum in complexity O(Nlog2N)
    fi.close();
    fo.close();
    return 0;
}
```