Binary Searching in the Presence of Comparison Errors

Bruno Carpentieri
Dipartimento di Informatica ed Applicazioni
Università di Salerno
84081 Baronissi (SA), Italy

Abstract

We consider the problem of identifying an unknown value $X \in \{a_1, \ldots, a_n\}$ using only "X ≤ C ?" queries, when at most E of the comparisons may receive erroneous answers.

We describe a strategy that solves this problem by using a number of comparisons that is close to the optimal. In fact we show we need less than $\log(n) + E\log(\log(n)) + \log(\log(n) + E\log(\log(n)) + O(\log(\log(\log(n))) + O(E\log(E))$ *comparisons, while the bound for the problem is* $\log(n) + E\log(\log(n)) + O(E\log(E))$.

Keywords Searching, Error, Lie, Game, Strategy, Error Correcting Codes.

1 Introduction

Let X be an unknown number which we wish to identify by asking "Yes-No" questions. Our goal is to minimize the number of questions required in the worst case, taking into account that some of the answers received may be erroneous. This problem comes in several versions based on the possible values of X and the nature of the "Yes-No" questions. We restrict the allowable questions to comparisons, i.e. questions of the form *"is $X \leq C$?"* where C is a specified integer element of the table we are searching in.

Similar problems have been faced before in literature. The original problem was proposed by Rényi in [1] and by Ulam [2] who asked "how many Yes-No questions are needed to identify an unknown number beetween 1 and one milion if at most 1 of the answers may be fault?". [4] solves the continuous version of the original problem. In [5] Pelc gives a solution of Ulam's problem for questions of the form "Is X in T?" were T is a discrete set. Recently there has been a growing interest in this problems and in its many facets. The state of the art of the research on Ulam problems and related aspects of fault tolerant binary search have been studied by Cicalese in [7] and by Pelc in [8].

Generalizing Ulam's problem, in this paper we consider that at most E, with $E \geq 1$, of the answers may be incorrect.

2 The Optimal Solution in the Continuous Case

In [4] the continuous version of this problem has been optimally solved by Rivest and others. In that case we search for a real number X in a half open normalized interval (0,1]. The solution of the continuous problem will represent the lower bound for the discrete case problem.

Given $\varepsilon > 0$ (for example $\varepsilon = 1/\mathrm{n}$ where n is an integer) and a hidden $X \in (0,1]$, in the continuous version of Ulam's problem we are required to minimize the number of comparisons needed to specify a subset of (0,1] of size $\leq \varepsilon$ that contains the unknown X.

Let $A = (A^0, A^1, \ldots, A^E)$ be an (E+1)-tuple such that $\mathrm{X} \in A^e$ iff exactly e of the previous answer are incorrect. A *state* of the problem will be a pair (q, A), where q is the number of comparisons remaining and $A = (A^0, A^1, \ldots, A^E)$. Following an idea of Berlekamp [3], Rivest et al. in [4] define the weight of a state A_q when there are q questions left, as:

$$w(q, A_q) = \sum_{e=0}^{E} (\binom{q}{E-e}) |A_q^e|,$$

where $(\binom{n}{m})$ denotes $\sum_{i=0}^{m} \binom{n}{i}$.

Let C be an element of the table we are searching in and $(q-1, Y)$ the state resulting from a positive answer to the comparison *"Is $X \leq C$?"*, let instead $(q-1, N)$ be the state resulting from a negative answer. In [4] is shown that if at each step of the algorithm the C of the next comparison is chosen so that $w(q-1, Y) = w(q-1, N)$, the number of comparisons is minimal.

It would seem that by choosing $\varepsilon = 1/n$ the discrete case could be approached. In fact using the same algorithm as in the continuous case, after $Q = \log(n) + E\log(\log(n)) + O(E\log(E))$ comparisons the size of the subinterval of the half open interval (0,1] containing X is less than $\varepsilon = 1/n$.

But even if we can reduce the subinterval containing X to less than $\varepsilon = 1/n$, the previous algorithm cannot easily be applied when the search space is discrete and of size n. In fact if $\{a_1, \ldots, a_n\}$ are the elements of the search table, we need to map them in (0,1]. There are essentially two possible solutions. The first is to map every $a_i \in \{a_1, \ldots, a_n\}$ in a point of (0,1].
For example:

$$f : \begin{cases} \{a_1, \ldots, a_n\} & \rightarrow & (0,1] \\ a_i & \rightarrow & f(a_i) = i(1/n) \\ & & i = 1 \ldots n \end{cases}$$

The second is to map every $a_i \in \{a_1, \ldots, a_n\}$ in a separate subset of (0,1].
For example:

$$g : \begin{cases} \{a_1, \ldots, a_n\} & \rightarrow & (0,1] \\ a_i & \rightarrow & g(a_i) = S_i \\ & & \ni' \; S_i \subset (0,1] \; and \\ & & S_i \bigcap S_j = \emptyset \\ & & i, j \in \{a_1, \ldots, a_n\}. \end{cases}$$

We also need f^{-1} and g^{-1} because when the algorithm in [4] chooses a $\overline{C} \in (0,1]$ to ask the next question, in the discrete case the comparison involved should be "$Is\ X \leq f^{-1}\overline{C}$?" in the first case or "$Is\ X \leq g^{-1}\overline{C}$?" in the second (in the discrete case we need to ask a question *"is $X \leq C$?"* where C is a specified integer element of the table we are searching in).

Since the algorithm produces $\overline{C} \in (0,1]$ often it happens that f^{-1} is not defined in that point. It is therefore necessary to define a new function $\overline{f}$ defined in (0,1] and such that $\overline{f}(i/n) = f^{-1}(i/n) = a_i$ $i = 1, \ldots, n$ and $\overline{f}(\overline{C}) = a_i$ if a_i is, between the n points of (0,1] chosen to represent the discrete interval, the one closer to $\overline{C} \in (0,1]$. This means that several $\overline{C} \in (0,1]$ lead to the same a_i, and therefore a_i may not always be the optimal choice. Similar problems arise for g^{-1}.

Furthermore, both using f or g, the problem of the correspondence between the final subinterval of (0,1] (whose size may be exactly $1/n$), and one and only one element of $\{a_1, \ldots, a_n\}$ arises.

3 A Fault Tolerant Binary Search Algorithm

Our goal is to define a search algorithm in a finite discrete set $\{a_1, \ldots, a_n\}$ that could work although a finite number of comparisons, at most E, may receive erroneous answers. We start, in analogy with [4], by defining a set A of subset of $\{a_1, \ldots, a_n\}$ as: $A = (A^0, A^1, \ldots, A^E)$, where $X \in A^e$ iff exactly e of the previous answer were incorrect.

We define a *state* of the problem as a couple (q, A_q) where q is the number of questions remaining and $A_q = (A_q^0, A_q^1, \ldots, A_q^E)$ where $X \in A^e$ iff exactly e of the previous answer were incorrect. Clearly we always have $\mathrm{X} \in \bigcup_{e=0}^{E} A_q^e$.

We define the *weight* of a state $S = (q, A_q)$ as $w(q, A_q) = \sum_{e=0}^{E} (\binom{q}{E-e} |A_q^e|$, were $\binom{n}{m} = \sum_{i=0}^{m} \binom{n}{i}$ and $|A|$ is the number of elements of the set A.

The algorithm is based on a recursive procedure. At any application of the main procedure, the initial number of comparison is chosen as $Q = min\{Q' || \log(w(Q', A))| \leq Q'\}$.

The basic idea of the algorithm is to choose the number C so that $|w_C(q-1, Y) - w_C(q-1, N)|$ is minimum for $C \in \{a_1, \ldots, a_n\}$ and so that $w_C(q-1, Y) < w(q, A_q)$ and $w_C(q-1, N) < w(q, A_q)$. In fact it is not always possible in this case to choose C so that $w_C(q-1, Y) = w_C(q-1, N)$, as in [4].

This choice could cause an unbalancement on the search tree of the algorithm and in case the answer is still ambiguous, i.e. if we have more than one $X \in \{a_1, \ldots, a_n\}$ candidate to be the solution, we will have to apply recursively the procedure on the new set A that represents the new initial situation.

Table 1 shows the pseudocode of the algorithm. Figure 1 and 2 shows the search tree produced by the algorithm when $n = 2^3$ and $E = 1$ and when $n = 2^4$ and $E = 1$.

4 Analysis of the Algorithm

We will first prove the correctness of the algorithm. Then we will analyze its worst case behaviour with respect to the number of comparisons.

Lemma 1 *Sufficient condition for the problem to be solved in a state $S = (q, A_q)$ is that $w(S) = 1$.*

Proof : $w(S) = 1$ implies $\exists i \ \ni' \ |A_q^e| = 0$ for $e = 0 \ldots E$, $e \neq i$, and $|A_q^i| = 1$. The solution will be the element of A_q^i. CVD.

We define a state $S = (q, A_q)$ *final* if $q = 0$ or if $\exists i \ \ni' \ |A_Q^i| = 0$ for $i = 0 \ldots E$, $i \neq j$ and $|A_q^j| = 1$.

We define the transition from a configuration $A_q = (A_q^0, A_q^1, \ldots, A_q^E)$ to a configuration $B_{q-1} = (B_{q-1}^0, B_{q-1}^1, \ldots, B_{q-1}^E)$ an **improvement** if $|A_q^i| = |B_{q-1}^i|$, for $i = 0 \ldots (j-1)$, and $|A_q^j| > |B_{q-1}^j|$.

Of course the number of possible **improvements** before reaching a state S such that $w(S) = 1$ is finite.

Lemma 2 *For every state $S = (q, A_q)$ not final, exists a C such that the comparison* "Is X ≤ C?" *produces an* **improvement**.

Proof : Let's suppose $|A_q^i| = 0$ for $i = 0 \dots (j-1)$. If $|A_q^j| \geq 2$ let $\overline{x}, \overline{y} \in A_q^j$ and $\overline{x} < \overline{y}$. A comparison *"Is $X \leq \overline{x}$?"* will lead to an **improvement** whatever the result is. Instead if $|A_q^j| = 1$ and $\overline{y} \in A_q^j$, there will be $E \leq k < j$ such that $|A_q^k| \geq 1$ and $h \in A_q^k$. Let $\overline{x} = min\{\overline{y}, h\}$, a comparison *"Is $X \leq \overline{x}$?"* will lead an **improvement** whatever the result is. CVD.

Lemma 3
$\exists Q \;\; \exists' \; Q = min\{Q' \mid \; |\log(w(Q', A))| \leq Q'\}$.

Proof : Of course $w(Q, A) \leq (\binom{Q}{E})n$ and $\sum_{i=0}^{n} \binom{n}{i} = 2^n$. The algorithm stops if $w(0, A_0) = 1$.
If we suppose that, after a comparison, the weight of the previous state is divided exactly in two, a Q that satisfies $(\binom{Q}{E})n < 2^Q$ satisfies also the thesis. Such a Q exists because for $Q > E$ the function 2^Q dominates the sum $(\binom{Q}{E})$.CVD.

Theorem 1 *The algorithm is correct.*

Proof : It follows from the lemmas 1, 2, 3 and the consideration that any comparison in the algorithm leads to an **improvement**. CVD.

We now analyze the worst case behaviour of the algorithm with respect to the number of comparisons.

Theorem 2 *Foreach state $S = (q, A_q)$, $w_C(q-1, Y) - w_C(q-1, N)$ is a non decreasing function of C.*

Proof. Let's suppose $S = (q, A_q)$. If C is such that $|w_C(q-1, Y) - w_C(q-1, N)|$ is minimum, the search table in the state (q, A_q) has been halved in two spaces $(q-1, Y)$ and $(q-1, N)$, resulting from the result of a comparison *"Is $X \leq C$?"*. A comparison with $\overline{C} < C$ would extend the search space resulting from a negative answer of the comparison and would decrease the space resulting from a positive one. So if $w_{\overline{C}}(q-1, \overline{Y})$ is the weight of the state subsequent to a positive result to the comparison *"is $X \leq \overline{C}$?"* and $w_{\overline{C}}(q-1, \overline{N})$ the state subsequent to a negative result, where $\overline{C} < C$ and C is such that $|w_C(q-1, Y) - w_C(q-1, N)|$ is minimum respect to C, then $w_{\overline{C}}(q-1, \overline{Y}) \leq w_C(q-1, Y)$ and $w_{\overline{C}}(q-1, \overline{N}) \geq w_C(q-1, N)$. Iterating this demonstration scheme for every $\overline{\overline{C}} < \overline{C} < C$ and applying it for every $\underline{\underline{C}} > \underline{C} > C$, we obtain the thesis. CVD.

Given n and E, xe define a numeric succession Q(i):
$Q(1) = \log(n) + E\log(\log(n)) + O(E\log(E))$.
$Q(i) = 0$ if $Q(i-1) = 0$ or if $\log(Q(i-1)+1) + E\log(\log(Q(i-1)+1)) + O(E\log(E)) < 1$.
$Q(i) = \log(Q(i-1)+1) + E\log(Q(i-1)+1) + O(E\log(E))$ in the other cases.

Theorem 3 *The number of comparisons in the algorithm RIC.ERR. is less than* $\sum_{i=1}^{\infty} Q(i)$.

Proof. At the first step of the algorithm, the is the initial number of comparisons Q is chosen so that $|\log(w(Q, A))| \leq Q$, where A is $A^0 = \{1 \dots n\}$ and $A^e = O, e = 1 \dots E$. The value of Q will be the same number of comparisons needed in the continuous case when $\varepsilon = 1/n$ and at least E comparisons may receive erroneous answers. Moreover the value of the weight $w(Q, A)$ will be the same as in the continuous case. In fact, for the Procedure RIC.ERR. in Table 1 in the initial situation:
$w(Q, A) = \sum_{e=0}^{E} (\binom{Q}{E-e}) |A_Q^e| = (\binom{Q}{E})n$.
Q will be chosen so that $|\log(w(Q, A))| \leq Q$, that is:
$(\binom{Q}{E})n \leq 2^Q \iff (\binom{Q}{E})2^{-Q} \leq 1/n$.
This is the condition under which Q was chosen in [1], to get to $\varepsilon = 1/n$, with at most E comparison errors. Hence, for any Q, if RIC.ERR. could exactly halve at each step the weight $w(Q, A)$, then less than $\log(n) + E\log(\log(n)) + O(E\log(E))$ comparisons would be needed to identify $X \in \{1 \dots n\}$. In general the algorithm splits $w(Q, A)$ in two parts whose difference is $|w_C(q-1, Y) - w_C(q-1, N)|$ (where C minimizes the difference $|w_y(q-1, Y) - w_y(q-1, N)|$ between all $y \in \{a_1, \dots, a_n\}$). To bound the unbalancing after any step of the algorithm, let's suppose that C is the minimum of the function $|w_{C'}(q-1, Y) - w_{C'}(q-1, N)| \; \forall \; C' \in \{a_1, \dots, a_n\}$ and that $w_C(q-1, Y) - w_C(q-1, N) \leq 0$, while $w_{C+1}(q-1, \overline{Y}) - w_{C+1}(q-1, \overline{N}) > 0$. Since the weight function considered is increasing with C, the worst case for the algorithm is always better than choosing C when $w_{C+1}(q-1, \overline{Y}) - w_{C+1}(q-1, \overline{N}) = 0$. (if we have $w_C(q-1, Y) - w_C(q-1, N) \geq 0$, we'll consider $C-1$ instead of $C+1$).
The umbalancing is then bounded from above by $(\binom{q}{E})$. In fact, suppose the state is (q, A_q) the query *"Is $X \leq C$?"*, receives a positive (negative) result that leads to a state $(q-1, A_{q-1})$, and *"Is $X \leq C+1$?"*, equally receiving a positive (negative) result with next state $(q-1, \overline{A_{q-1}})$ the difference between the weights of the two states will be:
$|w_{C+1}(q-1, \overline{A_{q-1}}) - w_C(q-1, A_{q-1})| = |\sum_{e=0}^{E} (\binom{q-1}{E-e})(|\overline{A_{q-1}^e}| - |A_{q-1}^e|)| < (\binom{q-1}{E})$

because at least two of the differences $(|\overline{A^e_{q-1}}| - |A^e_{q-1}|)$ will be equal to 1 in module (and of opposite sign), while the others will be 0. Since $(\binom{q}{E}) \leq 2^q$, for every state (q, A_q), the maximum weight unbalancing in the search tree in one step is bounded from above by 2^q. Consider now how the weight changes during the execution of the procedure ALG in Table 1, going from a state (q, A_q) to the next state $(q-1, A_{q-1})$. At the beginning of the algorithm, after choosing Q (that is $Q(1)$), we have:
$w(Q, A_Q) = (\binom{Q}{E})n$
after the first comparison:
$w(Q-1, A_{Q-1}) \leq w(Q, A_Q)/2 + 2^{Q-1}$,
and so on, until we get:

$$w(0, A_0) \leq w(Q, A_Q)/2^Q + \sum_{i=0}^{Q-1} 2^i/2^i < 1 + Q.$$

So, after the first call of the procedure ALG, the weight $w(0, A_0)$ is less than $Q + 1$. If we suppose that the first call to RIC.ERR. has left a set A_0 with $A^e_0 = O$ for $e = 1 \dots E$ and $A^0_0 = \{a_1 \dots a_{Q+1}\}$. The new call to ALG has now to solve a new search problem in a discrete set $\{a_1 \dots a_{Q+1}\}$, when as many as E of the comparisons may receive erroneous answers. Applying again the previous arguments, it is possible to show that the number of comparisons needed by ALG in this new situation will be exactly $Q(2)$ and that the weight $w(0, A_0) < Q(2)$. By iterative applications of the same arguments, we can show that the number of questions needed is upper bounded by the sum $\sum_{i=1}^{\infty} Q(i)$. CVD.

Therefore, if $Questions(n, E)$ is the number of comparisons needed by the algorithm RIC.ERR. to identify an element of the discrete set $\{a_1 \dots a_n\}$, when as many as E comparisons may have erroneous answers, we will have: $Questions(n, E) < \sum_{i=1}^{\infty} Q(i) \leq \log(n) + E\log(\log(n) + \log(\log(n) + E\log(\log(n)) + O(E\log(E))) + E\log(\log(\log(n) + E\log(\log(n)) + O(E\log(E)))) + O(\log(\log(\log(n))))$.

5 Conclusions and Future Work

We have considered the problem of identifying an unknown value $X \in \{a_1, \dots, a_n\}$ using only *"$X \leq C$?"* queries, when at most E of the comparisons may receive erroneous answers.

The strategy we have described solves this problem by using a number of comparisons that is: $Questions = \log(n) + E\log(\log(n)) + \log(\log(n) + E\log(\log(n)) + O(\log(\log(\log(n))) + O(E\log(E))$.

We are currently experimenting the algorithm for a number of errors $E >> 1$ and for large values of n.

References

[1] A. Rényi "On a problem of information theory", MTA Mat. Kut. Int. Kozl. 6B (1961).

[2] S. M. Ulam, "Adventures of a mathematician", Scribner's, N. Y. (1976).

[3] E.R. Berlekamp, "Block coding for the binary simmetric channel", *Error-Correcting Codes*, pp. 61-85. Wiley (1968).

[4] R. L. Rivest, A. R. Meyer, D.J. Kleitman, K. Winklmann, J. Spencer, "Coping with errors in bynary search procedures", *Journal of Computer and System Sciences*, 20 (1980).

[5] A. Pelc, "Solution of Ullam's problem on searching with a lie", *Journal of combinatorial theory*, A 44 (1987).

[7] F. Cicalese, "Reliable computation with unreliable information", Ph.D. Thesis, University of Salerno, (2001).

[8] A. Pelc, "Searching games with errors - fifty years of coping with liars", *Theoretical Computer Science*, 270, pp. 71-109, (2002).

Input: n *Number of elements of the search table*
E *Maximum number of errors*
A *set of subset of 1 ... n, at the beginning it will be A[0] = {1 ... n}, A[I] = Ø I=1 ... E*
Output: ANSWER *The hidden number*

TYPE ARR = Array [0..E] of Set of {1 ... n};

global variables S,D : ARR;
ANSWER,ANSW,QQ : INTEGER;

Procedure RIC.ERR.;
BEGIN ** RIC.ERR **
ALG(A);
OUTPUT('The solution is ',ANSWER);
END ** RIC.ERR **

Procedure ALG (O:ARR);
BEGIN ** ALG **
QQ := MIN {Q' | $|\log(w(Q', O))| \leq$ Q'}
TROUGH(QQ,O); IF (ANSW = 0) THEN BEGIN
IF NOT SOLUTION(S) THEN ALG(S)
END
ELSE IF NOT SOLUTION(D) THEN ALG(D);
END; ** ALG **

Procedure THROUGH (Q:INTEGER; A:ARR);
BEGIN ** THROUGH **
Choose I
$\ni'$ $|w_I(Q-1, Y_{A_{Q-1}}) - w_I(Q-1, N_{A_{Q-1}})| = \min \{|w_x(Q-1, Y_{A_{Q-1}}) - w_x(Q-1, N_{A_{Q-1}}|)\}$
with x $\in \{1 \ldots n\}$ and
$w_x(Q-1, Y_{A_{Q-1}}) < w(Q, A)$ and
$w_x(Q-1, N_{A_{Q-1}}) < w(Q, A)$.};
S := $Y_{A_{Q-1}}$;
D := $N_{A_{Q-1}}$;
ASK('X is $\leq$ ',I,' ?');
READLN(ANSW);
IF ANSW = 0 THEN BEGIN
IF Q-1 > 0 AND NOT(SOLUTION(S)) THEN THROUGH (Q-1,S);
END
ELSE IF Q-1 > 0 AND NOT(SOLUTION(D)) THEN THROUGH (Q-1,D);
END; ** THROUGH **

Function SOLUTION(A:ARR) : BOOLEAN;
BEGIN ** SOLUTION **
IF $\exists|$ X $\in \bigcup_{i=0}^{E}$ A[I] THEN BEGIN
ANSWER:=X;
SOLUTION:=TRUE
END
ELSE SOLUTION:=FALSE;
END ** SOLUTION **

Table 1: **The Algorithm RIC.ERR.**

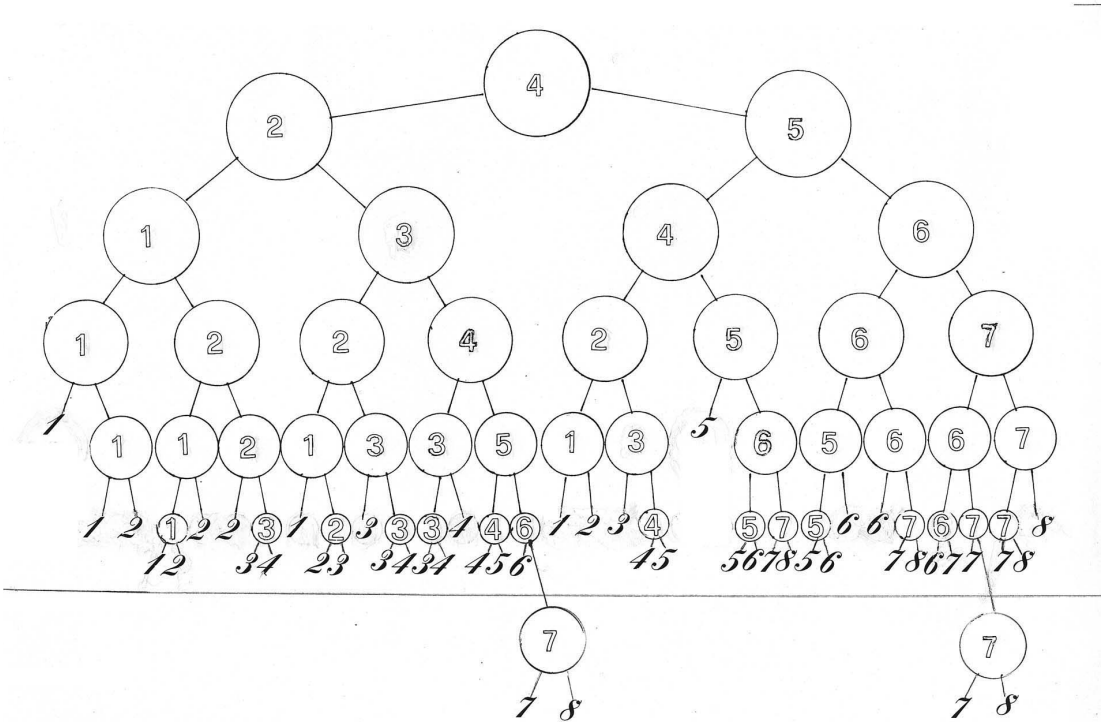

Figure 1: **The search tree for $n = 2^3$ and $E = 1$**

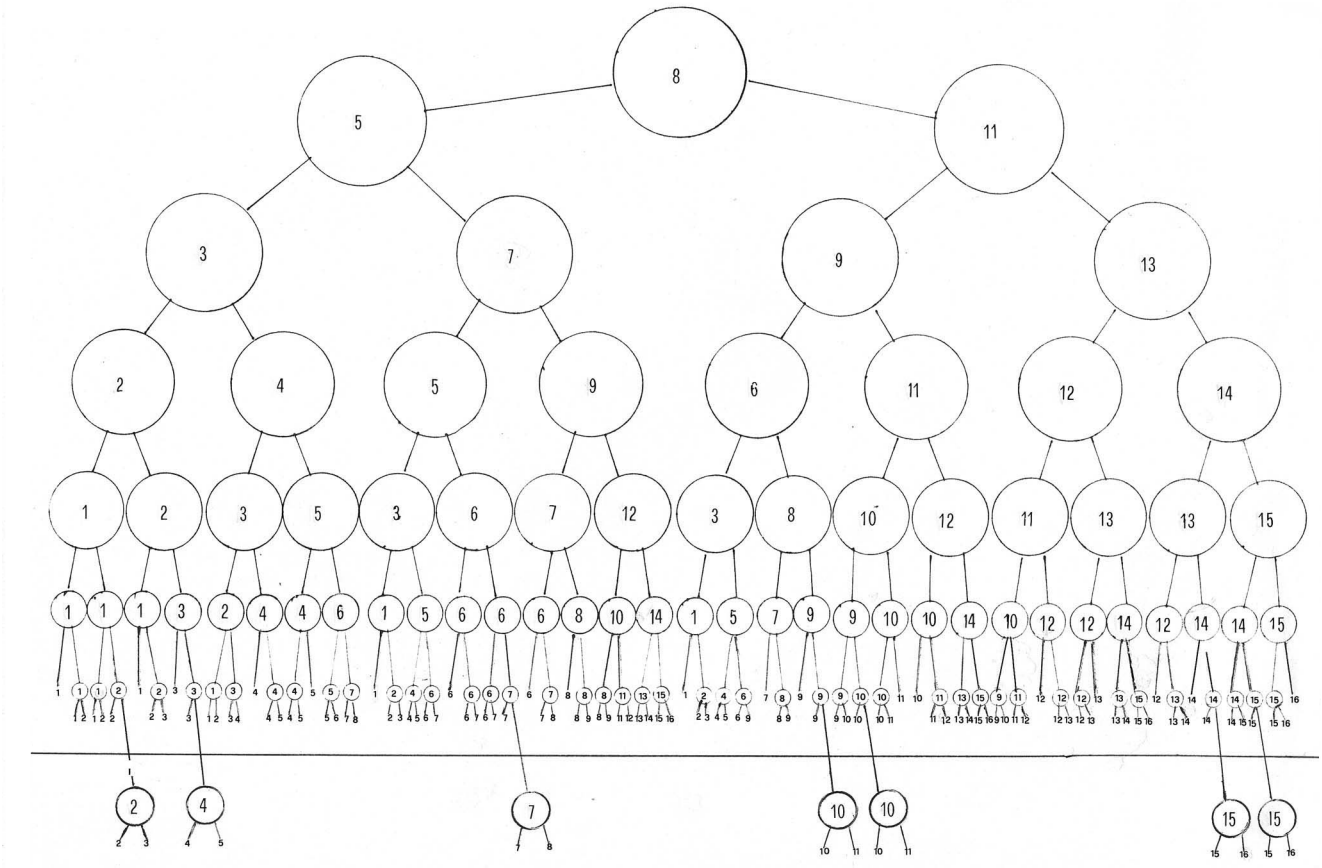

Figure 2: **The search tree for $n = 2^4$ and $E = 1$**