Given a read-only array containing $n$ integers where every integer occurs twice, except for two integers $a$ and $b$ that occurs only once.

How can you find $a$ and $b$ in $O(n)$ time using $O(1)$ storage?

Example: For input [13, 20, 8, 2, 13, 2, 17, 20] we want to find 8 and 17
Preparing for NCPC 2016

Jimmy Mårdell <yarin@spotify.com>
About me

- Studied at UmU 1997-2003
- Programming competition veteran
  - Won Swedish Ch 1999-2001
  - Won Google Code Jam 2003
  - 2nd place TopCoder Open 2003
  - 3rd place NCPC 2015
- Worked at Spotify since 2010
The Triangle

Find a path from top to bottom such that the sum is maximized
The Triangle

Find a path from top to bottom such that the sum is maximized
The Triangle - Recursion

```c
#include <stdio.h>

int rows;
int tri[100][100];

int max(int x, int y) {
    return x > y ? x : y;
}

int solve(int x, int y) {
    if (y == rows - 1)
        return tri[y][x];
    return tri[y][x] + max(solve(x, y+1), solve(x+1, y+1));
}

int main() {
    int i, j;
    scanf("%d", &rows);
    for(i = 0; i < rows; i++)
        for(j = 0; j <= i; j++)
            scanf("%d", &tri[i][j]);
    printf("%d\n", solve(0,0));
    return 0;
}
```

$2^{\#(\text{rows})}$ recursive calls == SLOW
The Triangle - Memoization

```c
#include <stdio.h>

int rows;
int tri[100][100];
int cache[100][100];

int max(int x, int y) {
    return x > y ? x : y;
}

int solve(int x, int y) {
    if (cache[y][x] >= 0)
        return cache[y][x];
    if (y == rows - 1)
        return tri[y][x];
    return cache[y][x] = tri[y][x] + max(solve(x, y+1), solve(x+1, y+1));
}

int main() {
    int i, j;
    scanf("%d", &rows);
    for(i = 0; i < rows; i++)
        for(j = 0; j <= i; j++) {
            scanf("%d", &tri[i][j]);
            cache[i][j] = -1;
        }
    printf("%d
", solve(0,0));
    return 0;
}
```
#include <stdio.h>

int max(int x, int y) {
    return x > y ? x : y;
}

int main() {
    int i, j, rows;
    int tri[100][100];
    scanf("%d", &rows);
    for(i = 0; i < rows; i++)
        for(j = 0; j <= i; j++)
            scanf("%d", &tri[i][j]);
    for(i = rows-2; i >= 0; i--)
        for(j = 0; j <= i; j++)
            tri[i][j] += max(tri[i+1][j], tri[i+1][j+1]);
    printf("%d\n", tri[0][0]);
    return 0;
}
Dissecting a programming problem
The four (five) phases of solving a problem

- Understand the problem
- Solve it
- Implement the solution
- Testing
- (Debugging)
Understanding the problem

- Length and complexity of text not necessarily an indicator of difficulty
- See beyond the story - what’s the underlying problem?
- Validate that you understand the problem by going through the examples
Solving the problem

- Mentally go through your algorithm toolbox
- Are there obvious subproblems?
- Input constraints is usually a good hint at what time complexity is required
- Validating ideas and prototyping
The implementation phase

• This is not software engineering…
• ..but use reasonable variable names and split up into functions/methods
• Test individual steps if large problem
• The choice of programming language is not that important
  • But be comfortable in the language you choose
  • No need to know intricate language details
  • But know its standard libraries! (data structures)
• Paper coding
Testing and debugging

- Maximum input case
- Corner cases
  - Empty or minimal input
- Be smart - test “just enough”
- Abusing the judge system
What should you practice on?
Know your environment and language

- Be comfortable reading input and format output
- Custom sort comparison function
- Nothing beats an IDE
- Know how to debug your code
  - Use IDE step-through-code and breakpoints
  - “printf”-debugging
  - GDB
- Array bound checking is a blessing
Language specific stuff

- C/C++:
  - Allocate arrays slightly larger than you think is necessary
  - STL! sort, set, map, lower_bound, upper_bound, random_shuffle, push_heap, next_permutation
- Java:
  - Allocate arrays based on input size
  - HashMap, TreeSet, BigInteger, Line2D
- Don’t use other languages :)
General tips and tricks

- NCPC usually contains 2-4 easy problems, 1-3 hard problems and the rest are “medium”
- Which problems have others solved?
- Skip geometry problems first 3 hours
- Look at input constraints and test data
- Iterate through your algorithm toolbox
- Brute force small cases
- “Offline” debugging
Algorithm and data structures

Breadth First Search

Heap

Trie

Topological Sorting

Euler Tour

Memoization

Interval Trees

Sort Algorithms

Network flow

Binary search

Strongly Connected Components

Depth First Search

Fenwick Tree

Line sweep

Nim Theory

Dynamic Programming

Dijkstra

Union Find

Knuth-Morris-Pratt

Gauss Elimination

2-SAT

Extended Euclides Algorithm

Line Intersection

Gauss Elimination
Practice, practice, practice

- Repetition is the mother of learning
  - If you’ve implemented binary search 100 times, you won’t screw up the next time.
- Practice sites:
  - Kattis (https://open.kattis.com/)
  - Google Code Jam (https://code.google.com/codejam)
  - TopCoder (http://www.topcoder.com/tc)
  - Codeforces (http://codeforces.com/)
KTH popup course

http://www.csc.kth.se/utbildning/kth/kurser/DD2458/popup15/uppgifter/

Vecka 2: Datastrukturer

- Almost Union-Find
- CD
- Chopping Wood (*)
- Movie Collection (*)
- The SetStack Computer
- Turbo

Vecka 3: Aritmetik

- Beat the Spread
- Dead Fraction (*)
- Ignore the Garbage
- Perfect Pth Powers (Tänk efter!)
- Three Digits
- Towers of Powers 2: Power Harder (*)

Vecka 4: Dynamisk programmering igen

- Cudak
- Downpayment
- Funny Games (*)
- Mountain Road (*)
- Pebble Solitaire
- The Uxuul Voting System

Vecka 5: Grafalgoritmer I

- Frogger
- George
- Getting Gold
- Island Hopping
- Killing Aliens in a Borg Maze (*)
- Proving Equivalences (*)
From theory to practice
Ambiguous Dates (NWERC 2001)

Given a string representing a date, determine all possible valid interpretations of the date. Year, month and date may be in any order, with or without leading zeros, with or without separators. Valid years are 1700-2299

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1631/02/29</td>
<td>Invalid</td>
</tr>
<tr>
<td>2001-11-03</td>
<td>2 valid interpretations</td>
</tr>
<tr>
<td>010203</td>
<td>36 valid interpretations</td>
</tr>
</tbody>
</table>
Chess x Tetris (NWERC 1999)

Determine if it’s possible to fit 16 given Tetris pieces onto an empty 8x8 board. The pieces may be rotated or flipped.
Backtracking

- Put a piece somewhere on the board
- Recurse
- Backtrack when there are no more options
- Question: Is this feasible?
  - Can we estimate the running time of such a solution?
Backtracking

Randomly placing the pieces is a bad idea… most pieces have more than 100 different placements (including rotation and flipping), will take forever.
Backtracking

Better idea: Fill up from bottom.
Good enough?
 MEME (SM 1998)

Given a boolean expression with a few variables (lowercase letters), determine all variable combination that causes the expression to be true (uppercase letter = variable is true, lowercase = variable is false).

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a &amp; b &amp; c</td>
<td>ABC</td>
</tr>
<tr>
<td>a &amp; (b</td>
<td>c)</td>
</tr>
<tr>
<td>(a</td>
<td>!a) &amp; (b</td>
</tr>
</tbody>
</table>
Basically Boolean (NWERC 1999)

Given a boolean expression with some variables, what variables affects the evaluation of the expression? The expression can contain parenthesis, logical operators (or, and, not, =>) and at most 10 variables.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A or B) and (C or D)</td>
<td>A B C</td>
</tr>
<tr>
<td>(A or not B) and (not A or not B)</td>
<td>B</td>
</tr>
<tr>
<td>((((B and C) or (D and B)) =&gt; (A or B))</td>
<td>-</td>
</tr>
</tbody>
</table>
Expression parsing

- Lot of known algorithms for doing efficient expression parsing, but…
- Keep it simple!!
- Reduce expression by search & replace is usually fine
private static Pattern p = Pattern.compile("(.*)\\(((^[^\\]*)\\)(.*)")

private static boolean evaluate(String expr) {
    while (true) {
        Matcher m = p.matcher(expr);
        if (m.matches()) {
            expr = m.replaceFirst("\$1" + (evaluate(m.group(2)) ? 1 : 0) + "\$3");
        } else {
            break;
        }
    }

    if (expr.length() == 1) return expr.equals("1");
    while (expr.contains("not ")) {
        expr = expr.replace("not 1", "0");
        expr = expr.replace("not 0", "1");
    }

    String[] split = expr.split("[\-\s]\d\s*\-\d\s*\-\d\s*");
    switch (split[1]) {
        case "and" : return evaluate(split[0]) && evaluate(split[2]);
        case "or" : return evaluate(split[0]) || evaluate(split[2]);
        case "->" : return !evaluate(split[0]) || evaluate(split[2]);
    }

    throw new RuntimeException("Error parsing " + expr);
}
Basically Boolean (NWERC 1999)

1. Evaluate expression for all possible variable assignments
2. To determine if a specific variable X is not affecting the expression:
   1. Check that for all other possible variable assignments, the entire expression is the same regardless of the value of X
Trying all subsets or permutations

Very common that you want to try “all subsets” or “all permutations” of a set.

- All subsets: Bitmask loop is convenient (see code snippet)
- All permutations: Recursion (or next_permutation if you use C++)

```java
int N = 10;

for (int mask = 0; mask < (1<<N); mask++) {
    for (int i = 0; i < N; i++) {
        if (((1<<i) & mask) > 0) {
            System.out.print(i + " ");
        }
    }
    System.out.println();
}
```
Currency (SM 1998)

Given the exchange rate between $M$ pairs of currencies ($M < 1000$), determine if it’s possible to earn money (arbitrage) by exchanging back and forth between different currencies. You should start and end with the same given currency.

3
SEK USD 8.00
USD DEM 0.60
DEM SEK 0.20
SEK
Borg (SM 2001)

Given a grid with the start position of the Borg and a number of aliens to assimilate, determine the minimal number of steps the Borg needs to walk to assimilate all aliens. At the start, or after assimilating an alien, the Borg can split into multiple groups.
Mushroom Misery (SM 2000)

Given up to 1000 circles of various sizes in a grid, determine the number of squares in the grid where some part of the grid square is inside a circle.

Grid size is at most 1,000,000 x 1,000,000.
String Factoring (SM 2000)

The string DOODOO can be factored to (DOO)$^2$, or, one step further, to (D(O)$^2$)$^2$. The length of a factored string is the number of letters in it, not parenthesis or exponents. Given an arbitrary string (80 characters at most), what’s the shortest factored representation of it?

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Explantation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRATTATTATTIC</td>
<td>6</td>
<td>PR(A(T)$^2$)$^3$IC</td>
</tr>
<tr>
<td>GGGGGGGGGGGG</td>
<td>1</td>
<td>(G)$^9$</td>
</tr>
<tr>
<td>PRIME</td>
<td>5</td>
<td>PRIME</td>
</tr>
<tr>
<td>BABBABABBABBA</td>
<td>6</td>
<td>(BAB)$^2$(A(B)$^2$)$^2$A</td>
</tr>
<tr>
<td>ARPARPARPARPAR</td>
<td>5</td>
<td>(ARP)$^4$AR</td>
</tr>
</tbody>
</table>
String Factoring (SM 2000)

1. If the string has a perfect factorisation \((X^Y)\), reduce it
   1. Repeat with new substring
2. Otherwise cut the string in two (try all possible cuts)
   1. Solve recursively for each piece
   2. Cache the result!
private static Map<String, String> memo = new HashMap<>();

private static String solve(String s) {
    if (s.length() == 1) return s;
    if (memo.containsKey(s)) return memo.get(s);

    String best = null;

    for (int rep = 1; rep <= s.length() / 2; rep++) {
        boolean works = true;
        for (int i = 0; i < s.length() && works; i++) {
            if (s.charAt(i) != s.charAt(i % rep)) {
                works = false;
            }
        }
        if (works) {
            String t = solve(s.substring(rep));
            if (best == null || t.length() < best.length()) best = t;
        }
    }

    for (int i = 1; i < s.length(); i++) {
        String t = solve(s.substring(0, i)) + solve(s.substring(i));
        if (best == null || t.length() < best.length()) best = t;
    }

    memo.put(s, best);
    return best;
}
Submarines (SM 2001)

Given up to 50 islands (polygons) and up to 100 submarines (lines), determine if the submarines are on land, on water or partially on land.
Line intersection & point-in-polygon

template <class P>
int lineIntersection(const P& s1, const P& e1, const P& s2, const P& e2, P& r) {
  if (((e1-s1).cross(e2-s2)) {  
    r = s2-(e2-s2)*(e1-s1).cross(s2-s1)/(e1-s1).cross(e2-s2);
    return 1;
  } else {
    return -((e1-s1).cross(s2-s1)==0 || s2==e2);
  }
}

template <class It, class P>
bool insidePolygon(It begin, It end, const P& p, bool strict = true) {
  int n = 0;
  for(It i=begin,j=end-1;i!=end;j=i++){
    if (onSegment(*i, *j, p)) return !strict;
    n += (max(i->y,j->y) > p.y && min(i->y,j->y) <= p.y && ((*j-*i).cross(p-*i) > 0) == (i->y <= p.y));
  }
  return n&1;
}
Lands Lords (NWERC 1999)

Given a division of land by up to 1000 straight lines, count the number of pieces of land. No three lines intersect at the same interior point.
Euler's Formula

In a connected planar graph, the following holds:

$\#\text{vertices} - \#\text{edges} + \#\text{faces} = 2$