GB21802 - Programming Challenges

Week 1 - Data Structures

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Science

2015-04-22,25

Last updated April 25, 2016

Summary for Week 0

- How the course works;
- How to solve simple (ad-hoc) programming challenges;
- Four simple programming challenges;

Results for Week 0

Problem	Total	Submissions	Solutions
Division of Nlogonia	31	27	27
Cancer or Scorpio	31	25	19
3n+1 Problem	31	23	21
Request for Proposal	31	18	13

Congratulations to all who submitted!

The problems for this week are already online.

Outline for Week 1

Week Theme: Data Structures

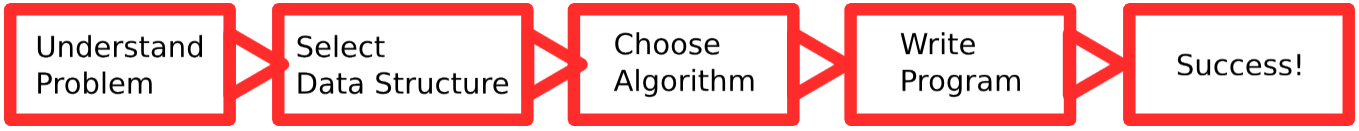
This week, we will review and discuss the different data structures that are often used in competitive programming problems.

- Types and Variations of Data Structures (Linear 2D, 3D, Dynamic, Non-linear)
- Discussion on Implementation and notes;
- Discussion about next week exercises;
- Bitmasks (Monday?)

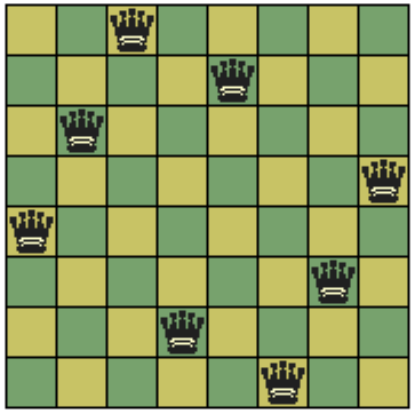
Motivation: Why study data structures?

- The correct data structure makes the algorithm faster;
  Example: As we saw last week, the solution space of the 8-queen problem depends on the chosen structure.

- A good implementation of a data structure makes the implementation easier;
  Exemple: Using a linked list, you raise the possibility of null pointer errors.



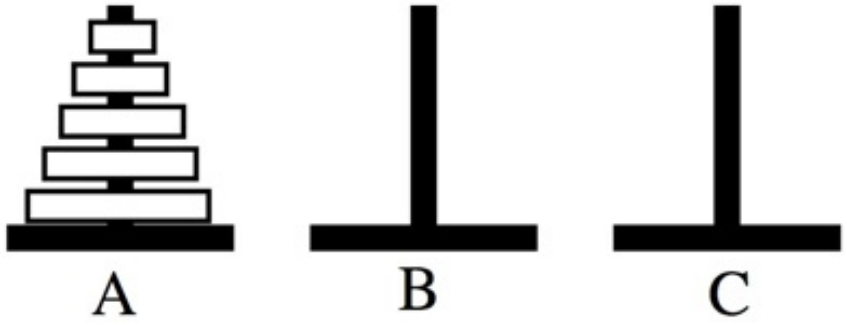
Example 1: 8 Queen Problem (UVA 750)



Given the initial position of one queen, how many correct solutions exist?

- X,Y position representation = 58^8 total solutions
- Column position representation = 8^8 total solutions
- Permutation representation = 8! total solutions

Example 2: The Towers of Hanoi



- You have N disks and K poles. Each disk has unique size s_i.
- A disk i can be moved from one pole to another.
- A move of disk i to pole k is only valid if k has no disks smaller than i
- Find the list of moves to move all disks from pole 1 to pole K.

How do you represent the data in this problem?

Another way to visualize the Towers of Hanoi

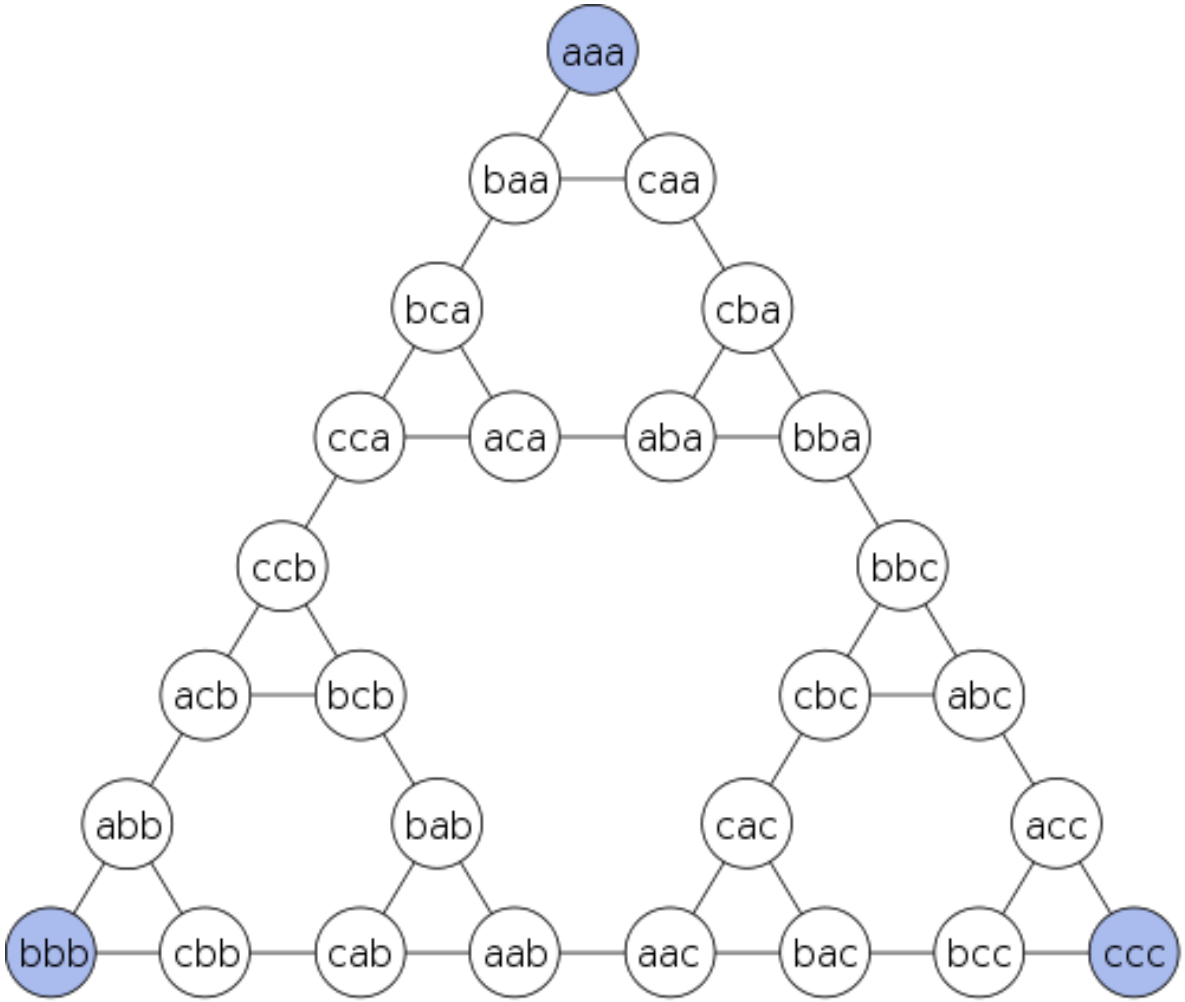
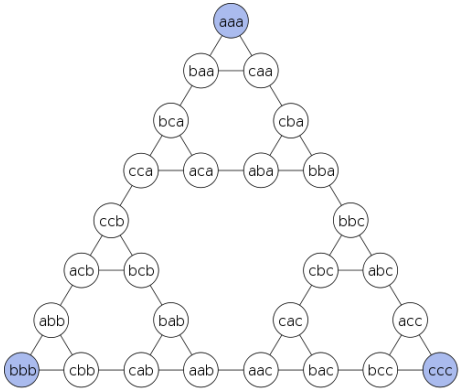


Image created by nonenmac

Explaining the Tower of Hanoi Data Structure

- Each node identifies an state in the problem;
- Each character in the string represents one disk and its position;
- We can have at most 3 state transitions at each state (can you prove it?)
- To solve the Towers of Hanoi problem, we find the path between the start and end states.
- (just beware of state explosion)



What are data structures?

A data Structure (DS) is a mean of storing and organizing data.

Different Data structures have different strengths. You need to have a general idea of those, in order to select and use the right DS for the right problem.

What do you need to know about DS?

Characteristics of DS

- Time Complexity: Insertion, Removal, Initialization, Access;
- Space Complexity: Repetitions of data, Empty Space;
- Others: Restrictions, Special behaviors;

Usage of DS

- Is the DS in the STL or Java API? How to call it?
- If not, how to implement the DS?
- How to perform basic operations on the DS? (Modification/Sort/Search)

As you learn new things, don't forget to expand your library

CAVEAT: When theory is important

When solving programming challenges, we will normally focus more on the practical aspects of Data Structures (how fast it is? how to use it?). It is not important to us to understant WHY they work as they do.

However, for a well rounded scientist, it is important to know both the practical and theoretical aspects of many techniques.

Linear Data Structures

One of the most common DS used in Programming Contests are.

Linear Data Structures

The elements can be ordered in a linear sequence:
left to right, top to down (Including multiple dimensions).

- Static Array:
  - The size is fixed, or estimated in advance.
  - Native data types. Make them bigger than necessary!
- Dynamic Array:
  - C++ STL vector / Java ArrayList;
  - Number of elements is truly unknown;
  - Learn how to use .iterators!

Specialized Linear DSs

- Array of Booleans (C++ bitset):
  Useful operations such as reset, set (initialization) and test (bounded access)
- Stack (C++ stack/Java stack):
  O(1) push and pop. Used for recursion/Postfix/Bracket matching
- Queue (C++ queue):
  push (to back), pop (from front), used for Breadth First Search, Topological Sorting
- Deque (C++ deque):
  push and pop from both ends. Used for "sliding window" algorithms.

Main operations in Arrays

When discussing linear DS, it is important to discuss two common actions done on them:

- Sorting – Changing the position of items in the array according to some ordering.
- Searching – Finding an item in an array with a certain value.

Actions on Linear DS – Sorting

Super formal Definition

Given unsorted stuff, sort them!

In programming contests, sorting is usually a preliminary step:

- sort a list to facilitate searching;
- sort a list to find highest values;
- sort a list for a greedy algorithm;
- sort a list to calculate cumulative values;
- etc...

Sorting Example – Vito's Family (UVA 10041)

Problem description

A gangster wants to move to a new house. The new house should have minimal distance from all his family, who live on the same street.

This program can be summarized as "find the median of the addresses and calculate the distance to all the houses."

```
for (i = 0; i < m; i++)
{
  cin >> add[i];
}
sort(add,add+m);
med = add[(m/2)];
```

Sorting algorithms

You probably have heard of a large number of sorting algorithms:

- $O(n^2)$ algorithms: Bubble, Selection, Insertion sort;
- $O(n\log n)$ algorithms: Merge/Heap/Quicksort;
- $O(n)$ algorithms: Bucket/Radix sort;

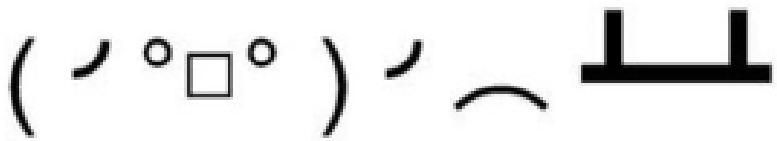
Can you remember the main characteristics of these algorithms?

How many sorting Algorithms are there?

- bubblesort
- insertion sort
- selection sort
- heapsort
- mergesort
- quicksort
- radix sort
- bin sort
- gnome sort
- library sort
- comb sort
- tree transversal
- sorting networks
- cocktail shaking sort
- bucket sort
- bogo sort
- bitonick sort
- ...
- And many more!

How many sorting Algorithms are there?

- bubblesort
- insertion sort
- selection sort
- heapsort
- mergesort
- quicksort
- radix sort
- bin sort
- gnome sort
- library sort
- comb sort
- tree transversal
- sorting networks
- cocktail shaking sort
- bucket sort
- bogo sort
- bitonick sort
- ...
- And many more!



Implementing Sorting

You don't need to know every one of those methods. At least for programming contest, you need a good idea of how to use sorting in your language's API

C++

Sorting is implemented by *STL algorithm*.
Eg: sort(memory_start,memory_end,function)
See also partial_sort() and stable_sort()

Java

To sort with Java we use "Collections.sort(array,function)"

Relax time

1. The song of the sorting people;

2. The quantum bogosort;

GB21802 - Programming Challenges

Week 1 - Data Structures

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Science

2015-04-22,25

Last updated April 25, 2016

Class 1B (Searching, Non linear data structures, Bitmasking)

Talking about the problems

Week 0: Don't forget to send the source code to Manaba
(Almost everyone did this already)

Week 1: Don't forget that the deadline is this week!

(I want you to rest during GW)

Week 1: Data Structures
Deadline: 3 days, 23:07 hours from now

#	Name	Sol/Sub/Total	My Status
1	Jolly Jumpers	14/15/32	
2	Army Buddies	4/8/32	
3	Rotated square	3/3/32	
4	File Fragmentation	1/1/32	
5	Contest Scoreboard	1/2/32	
6	Multitasking	1/1/32	
7	Jollybee Tournament	1/1/32	

Jolly Jumpers

Test a “jolly” sequence.

- Extremely easy problem, if you convert the input correctly;
- What information do we actually need from the input?

Army Buddies

Simulate changes in the data.

- Similar idea to Jolly Jumpers, but different data structure.
- What information do we need to store?
- What information do we need to change?

Rotated Square

Count the number of patterns.

- “bothersome” problem.
- What data to store?
- What is the program structure?

File Fragmentation

Find the original bit string.

- Understanding the problem: What are looking for here?
- Finding out the correct combination:
  - Method1: Elimination;
  - Method2: Counting;

Contest Scoreboard

Output winning teams in a competition.

- Actually the second easiest problem of the week. Make sure to read all problems once before starting!
- Problem is about sorting multiple categories.
  - You could create a single sort function to test all categories
  - Or you could sort the data multiple times (sort must be stable!)

Multitasking

Given a list of time intervals, find out if they overlap.

- Simple, but long problem
- First: Imagine the solution
- Second: What kind of dataset do you need?
- Caution 1: unecessary loops!
- Caution 2: interval ends may overlap;

Jollybee Tournament

Calculate the number of WO matches.

- Is the number of WO matches equal to the number of missing players?
- Does it matter who wins?
- Can we represent the problem in a non-tree, iterative manner?
  - Can we do it without iterating over each pair?

Searching in an array

Finding the highest value

- Unsorted array: $O(n)$
- Sorted array: $O(nlogn + 1)$

What if we need to repeat the operation k times?

Finding a specific value

- Unsorted array: $O(n)$
- Sorted array: $O(n\log n + \log n)$ – binary search!

But can you implement a sorted array quickly?

Binary Search in C++

You can use algorithm::lower_bound and algorithm::upper_bound

```
#include <iostream>
#include <algorithm>
#include <vector>
int main () {
  int myints[] = {10,20,30,30,20,10,10,20};
  std::vector<int> v(myints,myints+8);
  std::sort (v.begin(), v.end());

  std::vector<int>::iterator low,up;
  low=std::lower_bound (v.begin(), v.end(), 20);
  up= std::upper_bound (v.begin(), v.end(), 20);

  std::cout << "lower at " << (low- v.begin()) << '\n';
  std::cout << "upper at " << (up - v.begin()) << '\n';

  return 0; // up and low are memory indexes.
}
```

Non Linear Data Structures

Balanced Binary Search Tree

- C++ Map/Set (implemented as RB tree)
- Java TreeMap/TreeSet

Access/Deletion/Search in $O(\log n)$.

Heap

- C++ priority_queue
- Java PriorityQueue

Gets the highest value individual in $O(1)$

Tree Linear Structures: Example Problem

UVA 10226 – Hardwood Species

Input is up to 1.000.000 trees, up to 10.000 types. Sort and output the speciation. Test cases are not limited.

Because of the huge numbers, storing the trees in an array will not be enough. We need a fast data structure to solve this problem in time.
Otherwise the problem is not actually difficult.

Using Bitmasks

A bitmask is a lightweight version of a bitset. You can use an unsigned integer or an unsigned long directly as a proxy for a bitset.

```
#include <iostream>

using namespace std;

int main() {
    unsigned long bb = 3432;
    unsigned long kk = 14;
    cout << (bb & kk) << endl;
    cout << (bb << 3) << endl;
}
```

In a programming contest, they are very useful for quickly manipulating sets (specially if you need to modify a large number of members at the same time).

When to use bitmasks

In programming challenges, bitmasks are useful for quickly manipulating sets of items. Specially if you need to modify multiple sets at the same time.

$S = 34 = 100010$

Can be seen as a set with elements 1 and 5 present.
The index increase with the digit significance.

In regular programming, bitmasks are often used to quickly set/check flags or options.

Bitmasks in regular programming

- Parameter setting/testing

```
Gdx.gl.glClear( GL20.GL_COLOR_BUFFER_BIT |
                GL20.GL_DEPTH_BUFFER_BIT );
```

- Collision/Filtering in computer graphics

```
if( isFilled(sprite1Pixel) && isFilled(sprite2Pixel))
    return true;
```

Binary Operatons on Bitmasks (2)

- Multiply/Divide an integer by two :: shift bits left, right

```
S           = 34        =  100010
S = S << 1 = S*2 = 68 = 1000100
S = S >> 2 = S/4 = 17 =   10001
S = S >> 1 = S/2 =  8 =    1000
```

- To check if the ith item is on the set, use bitwise AND operation, (T = S & (1 « j)) and test if the result is not zero.

```
S           = 34        =  100010
j = 3, 1 << j           =  001000
i = 1, 1 << 1           =  000010
                           ------
Tj= S & ( 1 << j)       =  000000  = 0 # 3 is not set
Ti= S & ( 1 << i)       =  000010 != 0 # 1 is set
```

Binary Operations on Bitmasks (2)

- To set/turn on the jth item, use bitwise OR operation S |= (1 « j)

```
S              = 34          =  100010
j = 3, 1 << j                =  001000
                                ------ OR (S |= 1 << j)
S              = 42          =  101010
```

- To set/turn off the jth item, use bitwise AND operation S &= (1 « j)

```
S              = 50          =  110010
j = (1<<5)|(1<<3)            =  101000 # unset items 5,3
~j                           =  010111
                                ------
S &= ~(j)                    =  010010 # 18
```

Have some code with the previous examples!

```
#include <iostream>
using namespace std;

int main() {
    unsigned int S = 34;
    cout << (S<<1) << endl;
    cout << ((S<<1)>>2) << endl;
    cout << (((S<<1)>>2)>>1) << endl << endl;
    cout << (S & (1 << 3)) << endl;
    cout << (S & (1 << 1)) << endl << endl;
    cout << (S | (1 << 3)) << endl;
    S = 50;
    cout << (S & ~((1 << 5)|(1<<3))) << endl;
}
```

Bitsets

Bitsets can also be used in place of integer bitmasks:

```
#include <bitset>
#include <iostream>

using namespace std;

int main() {
    bitset<12> b(3432);
    cout << "3432 in binary is " << b << endl;

    bitset<12> k(14);
    cout << (b & k) << endl;
    cout << (b<<3) << endl;
}
```

Bitsets and Bitmasks

Most bitmask operations are also supported by bitsets. (But you can't mix them!)

- All binary operations are supported;
- Bitsets are of fixed size, but not limited to 32 and 64 bits.
- Bitset standard output is a bit string, not a decimal (but outputing a decimal is possible);
- But maybe bitmasks are more efficient?

More information about bitmasks

`http://www.drdobbs.com/the-standard-librarian-bitsets-and-bit-v/184401382`

Conclusion

Today we talked about:

- Searching using standard tools: Binary search, Best N search;
- Tree data structures for efficient searching;
- More about Bitmasks/Bitsets

Thanks for your time!