Competitive Programming

Frank Takes

LIACS, Leiden University

`https://liacs.leidenuniv.nl/~takesfw/CP`

Lecture 3 — Problem types

Recap

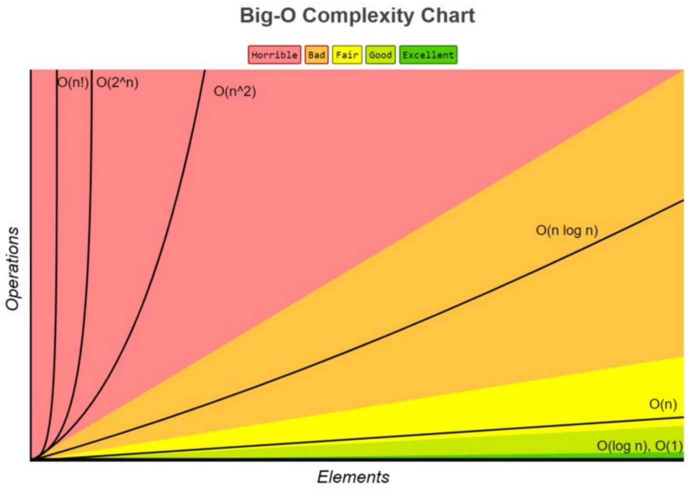
After (the) last (two) week(s) . . .

- you are registered for the course,
- familiar with competitive programming and the practical skills in C or C++ to participate in a programming contest,
- refreshed your knowledge of data structures and libraries,
- refreshed your knowledge of the C++ standard library,
- are able to use the Kattis programming contest platform in addition to DOMjudge,
- have practiced with various problems related to search, sorting and simulation.

Data structures and libraries (week 2)

- Linear structures: `array`, `vector`, `bitset`, `list`, `stack`, `queue`, `deque`
- Nonlinear structures: `priority_queue`
- Mapping and “hashing”: `set`, `map`, `multimap`, `multiset` (and `unordered_map`, `unordered_set`, etc.)
- Functions: `min`, `max`, `sort`, `binary_search`, `lower_bound`, etc.

Time complexity



Source: `http://bigocheatsheet.com`

Input size vs. complexity

Input size	**Expected time complexity**
$n \leq 10$	$O(n!)$ or $O(n^6)$
$n \leq 20$	$O(2^n)$
$n \leq 100$	$O(n^4)$
$n \leq 400$	$O(n^3)$
$n \leq 10^3$	$O(n^2 \log n)$
$n \leq 10^4$	$O(n^2)$
$n \leq 10^6$	$O(n \log n)$
$n \leq 10^8$	$O(n)$, $O(\log n)$ or $O(1)$

- Input size is usually $\leq 10^6$ due to I/O constraints

Problem types

- Sorting
- Searching
- Brute-force and backtracking
- Simulation
- Greedy
- Graphs
- Divide and conquer
- Dynamic programming
- String processing
- Geometry
- Mathematics

Simulation

- Also called ad-hoc
- General idea: the solution can be found by just programming whatever the problem description asks you to do
- Example: Week 2 Problem E - (Adding Words); you just needed a `map<string,int>`
- Tricky part (if any) is usually in edge cases
- Usually just ACCEPT or WRONG ANSWER,
  TIME LIMIT EXCEEDED is rare

Greedy

- General idea: solve using some locally optimal decisions
- Sometimes requires some sorting
- Example: *Coin change*. Given a target amount V in cents and a list of coin denominations $(1, 2, 5, 10, 20, 50)$, what is the minimum number of coins needed to represent amount V?

Greedy

- General idea: solve using some locally optimal decisions
- Sometimes requires some sorting
- Example: *Coin change*. Given a target amount V in cents and a list of coin denominations $(1, 2, 5, 10, 20, 50)$, what is the minimum number of coins needed to represent amount V?
- Solution: repeatedly select the largest denomination which is not greater than the remaining amount, and count the number of coins

Greedy

- General idea: solve using some locally optimal decisions
- Sometimes requires some sorting
- Example: *Coin change*. Given a target amount V in cents and a list of coin denominations $(1, 2, 5, 10, 20, 50)$, what is the minimum number of coins needed to represent amount V?
- Solution: repeatedly select the largest denomination which is not greater than the remaining amount, and count the number of coins
- Greedy can be difficult to recognize; use complete search or DP if unsure and the input size constraints allow it
- Examples:

Greedy

- General idea: solve using some locally optimal decisions
- Sometimes requires some sorting
- Example: *Coin change*. Given a target amount V in cents and a list of coin denominations $(1, 2, 5, 10, 20, 50)$, what is the minimum number of coins needed to represent amount V?
- Solution: repeatedly select the largest denomination which is not greater than the remaining amount, and count the number of coins
- Greedy can be difficult to recognize; use complete search or DP if unsure and the input size constraints allow it
- Examples: Dijkstra's algorithm, but also Kruskal's and Prim's algorithm for creating a minimal spanning tree of a weighted graph

Minimal spanning tree

- A **spanning tree** is a tree and subgraph of a given graph that covers all nodes of the graph
- In weighted graphs, a **minimal** spanning tree is one of minimal edge weight

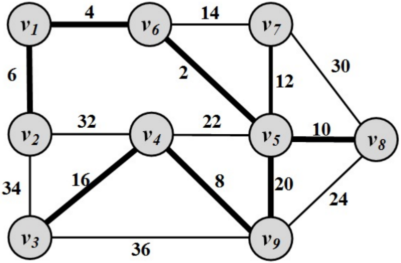
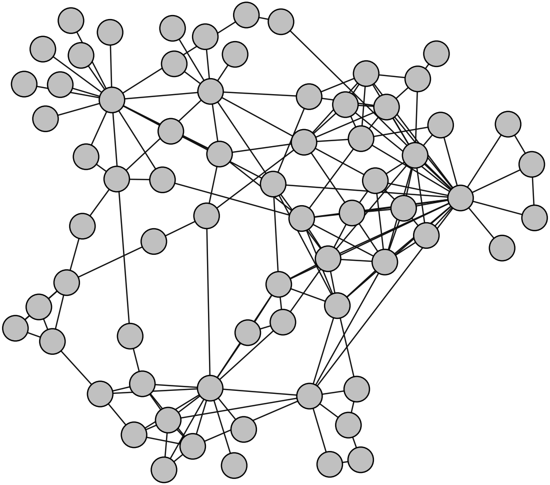


Image: Zafarani et al., Social Media Mining, 2014.

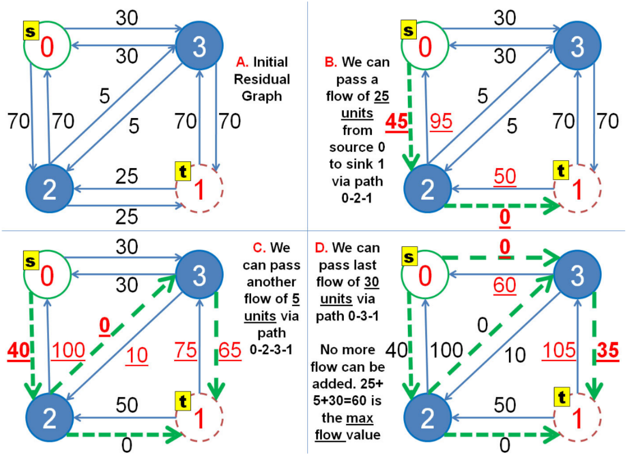
Graphs



Graphs

- Traversal: DFS, BFS, SSSP, APSP, Floyd-Warshall
- Weakly connected components: flood fill
- Strongly connected components: Kosaraju's / Tarjan's algorithm
- Articulation points and bridges (increase component count when removed)
- Directed acyclic graph (dag); can be sorted topologically
- Bipartite graphs; certain problems are no longer in NP
- Trees; no cycles, vertices $=$ edges $+ 1$

Graph flow



A. Initial Residual Graph

B. We can pass a flow of 25 units from source 0 to sink 1 via path 0-2-1

C. We can pass another flow of 5 units via path 0-2-3-1

D. We can pass last flow of 30 units via path 0-3-1

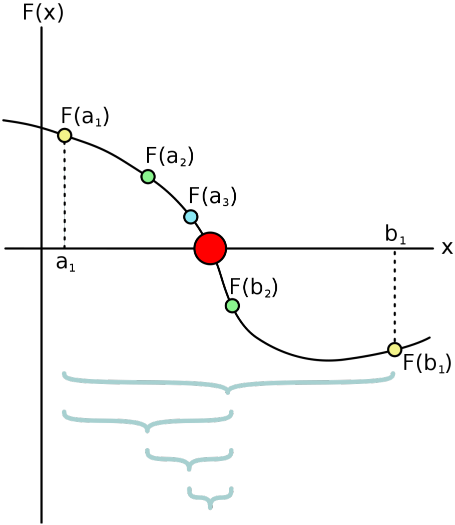
No more flow can be added. 25+5+30=60 is the max flow value

Divide and conquer

- Applicable when subproblems are independent of each other
- Not often encountered directly in a contest problem
- Implicitly part of sorting algorithms and various data structures
- Binary search is the most common application
- Bisection method: assess for some nontrivial function $F(x)$ for what value a certain optimum $F(x) = y$ is reached by refining a range $[a..b]$ using binary search until $F((a+b)/2) = y$

Bisection method

- Bisection method: assess for some nontrivial function $F(x)$ for what value a certain optimum $F(x) = y$ is reached by refining a range $[a..b]$ using binary search until $F((a+b)/2) = y$



Dynamic programming

- Dynamic programming (DP)
- Problems not solvable by
    - greedy approaches, because locally optimal decisions are insufficient and give WRONG ANSWER
    - exact search is too slow; TIME LIMIT EXCEEDED
    - divide and conquer is not applicable as the subproblems are not independent
- Main idea: build final solution from solution to subproblems
- Top-down approach: recursively compute the final solution and use memoization to avoid double work
- Bottom-up approach: start by solving subproblems and increase their “scope” until full problem is solved

Top-down DP

```
long long fib(long long n) {
    if(n == 0 || n == 1)
        return n;
    return fib(n-1) + fib(n-2);
} // fib (recursive)
```

Top-down DP

```
long long fib(long long n) {
    if(n == 0 || n == 1)
        return n;
    return fib(n-1) + fib(n-2);
} // fib (recursive)
long long fibs[43] = {0};

long long fib_topdown_dp(long long n) {
    if(n > 1 && fibs[n] == 0)
        fibs[n] = fib(n-1) + fib(n-2);
    return fibs[n];
} // fib in O(n) space

int main() {
    fibs[1] = 1;
    cout << fib_topdown_dp(40) << endl;
    return 0;
} // main
```

Bottom-up DP

```cpp
long long fib_bottomup_dp(long long n) {
    if(n == 0 || n == 1)
        return n;
    long long a = 0;
    long long b = 1;
    long long c;
    for(int i=2; i<=n; i++)  {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
} // fib in O(1) space

int main() {
    cout << fib_bottomup_dp(40) << endl;
    return 0;
} // main
```

String processing

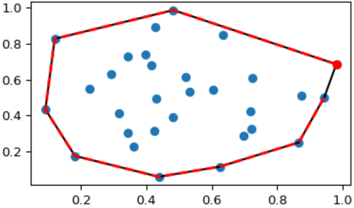
- `This is an input string.`
- (Re-)familiarize yourself with `<string>` functions
- Common problems:
  - encoding and decoding
  - frequency counting
  - parsing input
  - string comparison
  - string matching: given a string `T` of length n, find `S` of length m remember how Knuth-Morris-Pratt does this in $O(m+n)$?
  - string alignment: edit distance, etc.

Geometry

- Operations on points, polygons, circles and triangles
- Example problems: geometric distance, convex hull, line crossing
- Know your "`soscastoa`", π and functions in `<math.h>`

```
struct Point {
    double x, y;

    bool operator < (point b) const {
        if (fabs(x - b.x) > EPS)
            return x < b.x;
        return y < b.y;
    }
};
```



Mathematics

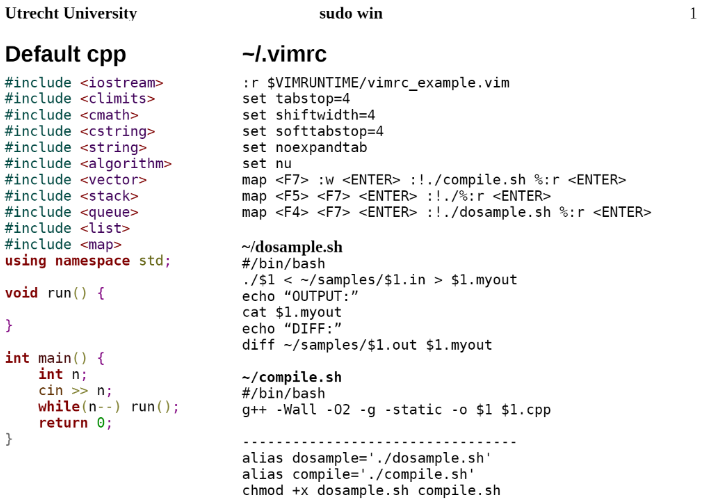
- General idea: solve a mathematical "puzzle"
- Sometimes, after solving the puzzle, the problem is trivial
- Often the mathematics is part of a larger solution
- Number theory: prime numbers, prime factors, factorial, modulo
- Combinatorics: Fibonacci numbers, binomial coefficients, Catalan numbers
- Big integers: GCD, modulo, base conversion; use Java or BigInteger class in C++

Programming contests

Skills for being competitive

- `using namespace std;` (or not)
- Pragmatic programming (when to stop optimizing?)
- Typing speed
- Finding bugs
- Writing extra test cases
- Know your language manual
- Team manuals
- Shorthand code

Team manual (example)



Utrecht University   sudo win   1

Default cpp

```cpp
#include <iostream>
#include <climits>
#include <cmath>
#include <cstring>
#include <string>
#include <algorithm>
#include <vector>
#include <stack>
#include <queue>
#include <list>
#include <map>
using namespace std;

void run() {

}

int main() {
    int n;
    cin >> n;
    while(n--) run();
    return 0;
}
```

~/.vimrc

```
:r $VIMRUNTIME/vimrc_example.vim
set tabstop=4
set shiftwidth=4
set softtabstop=4
set noexpandtab
set nu
map <F7> :w <ENTER> :!./compile.sh %:r <ENTER>
map <F5> <F7> <ENTER> :!./%:r <ENTER>
map <F4> <F7> <ENTER> :!./dosample.sh %:r <ENTER>
```

~/dosample.sh

```
#/bin/bash
./$1 < ~/samples/$1.in > $1.myout
echo “OUTPUT:”
cat $1.myout
echo “DIFF:”
diff ~/samples/$1.out $1.myout
```

~/compile.sh

```
#/bin/bash
g++ -Wall -O2 -g -static -o $1 $1.cpp

---------------------------------
alias dosample='./dosample.sh'
alias compile='./compile.sh'
chmod +x dosample.sh compile.sh
```

Source (newer version): `https://github.com/ludopulles/tcr/blob/master/tcr.pdf`

Shorthand code

```
#define REP(i,n) for(int i=0;i<(n);i++)

#define vi vector<int>

// or:

typedef long long int ll
typedef long double ld
```

- Usually added on top of a team's "solution template"

Collaboration in live contests

- Establish problem types and difficulty
- Assign problems to people
- One computer; use it wisely
- Focus moments for progress discussion
- Communication
- Printing
- Teams of two or three students

Lab session today and next week

- Discuss: python and language manuals
- Problems of this week linked on website
- Week after that: “soft contest” at 13:15

Credits

This course, in particular these slides, are largely based on:

- Antti Laaksonen, *Guide to Competitive Programming*, Springer, 2017.
- Steven Halim and Felix Halim, *Competitive Programming 3*, Lulu.com, 2013.
- T-414-AFLV: A Competitive Programming Course, `https://github.com/SuprDewd/T-414-AFLV`

Where applicable, full credit for text, images, examples, etc. goes to the authors of these books.