Algorithms Lab

September 26, 2012

Outline

We assume that you ...

- know how to code
- know the basic algorithms
- are able to turn a reasonably detailed idea into a working code! (otherwise the hints we give will not be very useful)
- are familiar with the $\mathcal{O}$ notation and the idea of estimating the runtime of your program based on the input size

Outline

1. C++ STL
2. Exercise hints

STL

- Coding data structure from scratch - painful and error prone
- STL - *map*, *set*, *queue*, *priority queue*
  - map - $\mathcal{O}(\log n)$ lookups
  - set - $\mathcal{O}(\log n)$ set membership tracking
  - queue - FIFO
  - priority queue - $\mathcal{O}(\log n)$ retrieval of an element with the highest priority, insertion
- look at the documentation

```
#include <algorithm>
...
vector<int> a;
...
sort(a.begin(), a.end());
```

```cpp
#include <algorithm>
...
struct Data {
  int key;

  bool operator<(const Data &a) const {
    return key < a.key;
  }
};
...
vector<Data> a;
...
sort(a.begin(), a.end());
```

Set

Same applies for set, map and priority queue

```
#include <set>
...
struct Data {
  int key;

  bool operator<(const Data &a) const {
    return key < a.key;
    }
};
...
set<Data> a;
...
set<Data>::iterator itr = a.find(data);
```

Outline

Exercise 1 - False coin

Two approaches

- (slower) Guess a solution
  - assume a coin is heavier (or lighter)
  - check consistency with the input
  - exactly one such coin – we found a solution
- (faster) Keep the track of possibly false coins

Exercise 1 - False coin

Two approaches

- (slower) Guess a solution
  - assume a coin is heavier (or lighter)
  - check consistency with the input
  - exactly one such coin – we found a solution
- (faster) Keep the track of possibly false coins

Exercise 1 - False coin

Two approaches

- (slower) Guess a solution
  - assume a coin is heavier (or lighter)
  - check consistency with the input
  - exactly one such coin – we found a solution
- (faster) Keep the track of possibly false coins

Two approaches

- (slower) Guess a solution
  - assume a coin is heavier (or lighter)
  - check consistency with the input
  - exactly one such coin – we found a solution
- (faster) Keep the track of possibly false coins

Exercise 1 - False coin

Two approaches

- (slower) Guess a solution
  - assume a coin is heavier (or lighter)
  - check consistency with the input
  - exactly one such coin – we found a solution
- (faster) Keep the track of possibly false coins

Exercise 1 - False coin

Two approaches

- (slower) Guess a solution
  - assume a coin is heavier (or lighter)
  - check consistency with the input
  - exactly one such coin – we found a solution
- (faster) Keep the track of possibly false coins

Exercise 1 - False coin - Summary

Trick learned

- guess a solution
- guess an interval in which solution lies, properties of a solution, etc.

Exercise 2 - Race Tracks

- Smallest number of hops
- Recall – BFS finds the smallest number of steps from vertex v_1 to v_2
- Model the input as a graph

Exercise 2 - Race Tracks

Naive model

- vertices - points in the grid
- edge between two vertices – hop from one point to the other
- hop depends on our current velocity
- different velocities give different hops from the same vertex
- graph with edges which dynamically change? doesn't sound good ...

Naive model

- vertices - points in the grid
- edge between two vertices – hop from one point to the other
- hop depends on our current velocity
- different velocities give different hops from the same vertex
- graph with edges which dynamically change? doesn't sound good ...

Exercise 2 - Race Tracks

Better model

- vertex is a pair (p, v) – p is a point in the grid, v is the velocity which we had upon arriving to that point
  - Start vertex – $(s, (0, 0))$
  - End vertices – $(t, (i, j))$, for all $-3 \leq i, j \leq 3$
- Number of vertices – at most $(30 \times 30) \times (7 \times 7) \approx 45000$
- Degree of each vertex at most 9 – at most 405000 edges
- Edges can be deduced from the "label" of a vertex – no need to store the whole graph explicitly

Better model

- vertex is a pair (p, v) – p is a point in the grid, v is the velocity which we had upon arriving to that point
  - Start vertex – $(s, (0, 0))$
  - End vertices – $(t, (i, j))$, for all $-3 \leq i, j \leq 3$
- Number of vertices – at most $(30 \times 30) \times (7 \times 7) \approx 45000$
- Degree of each vertex at most 9 – at most 405000 edges
- Edges can be deduced from the "label" of a vertex – no need to store the whole graph explicitly

Exercise 2 - Race Tracks

Better model

- vertex is a pair (p, v) – p is a point in the grid, v is the velocity which we had upon arriving to that point
  - Start vertex – $(s, (0, 0))$
  - End vertices – $(t, (i, j))$, for all $-3 \leq i, j \leq 3$
- Number of vertices – at most $(30 \times 30) \times (7 \times 7) \approx 45000$
- Degree of each vertex at most 9 – at most 405000 edges
- Edges can be deduced from the "label" of a vertex – no need to store the whole graph explicitly

Prerequisites

- BFS!

Trick learned

- Vertices of a graph can be represented with more complex objects than just numbers from 1 to n
- No need to store the graph explicitly in order to perform a BFS or similar algorithm

Exercise 3 - Forced Landing - First approach

Observations

- Equivalent – given an $h \times w$ matrix with $0/1$ elements, find an $r \times s$ submatrix with all 0 elements
  - up to a "constant" – we need a $(r-1) \times (s-1)$ submatrix, etc.
- r and s are fixed – top-left corner completely describes the rectangle

Exercise 3 - Forced Landing - First approach

Observations

- Equivalent – given an $h \times w$ matrix with $0/1$ elements, find an $r \times s$ submatrix with all 0 elements
    - up to a "constant" – we need a $(r-1) \times (s-1)$ submatrix, etc.
- r and s are fixed – top-left corner completely describes the rectangle

Exercise 3 - Forced Landing - First approach

Observations

- Equivalent – given an $h \times w$ matrix with $0/1$ elements, find an $r \times s$ submatrix with all 0 elements
  - up to a "constant" – we need a $(r-1) \times (s-1)$ submatrix, etc.
- r and s are fixed – top-left corner completely describes the rectangle

Exercise 3 - Forced Landing - First approach

Easy solution

- try all possible places for the top-left corner
- check whether the induced rectangle is empty (has all 0 elements)
- complexity
  - $\mathcal{O}(h \cdot w)$ possible rectangles
  - checking whether it is empty by simply iterating through all its elements – $\mathcal{O}(r \cdot s)$
  - in total $\mathcal{O}(hwrs)$ – too slow

Easy solution

- try all possible places for the top-left corner
- check whether the induced rectangle is empty (has all 0 elements)
- complexity
  - $\mathcal{O}(h \cdot w)$ possible rectangles
  - checking whether it is empty by simply iterating through all its elements – $\mathcal{O}(r \cdot s)$
  - in total $\mathcal{O}(hwrs)$ – too slow

Exercise 3 - Forced Landing - First approach

- try to solve a simpler problem first – suppose that $h = r = 1$
- an array instead of a matrix

Prefix sums

Given an array A, set $s[i] := \sum_{j=1}^{i} A[i]$. Additionally, set $A[0] := 0$.

- subarray $A[i], \ldots, A[i+s-1]$ is empty $\leftrightarrow$ $A[i+s-1] - A[i-1] = 0$
- apply similar trick to a matrix
  - reduces the complexity of checking whether a rectangle is empty from $\mathcal{O}(rs)$ to $\mathcal{O}(1)$!
  - in total $\mathcal{O}(hw)$ – works well for small and medium testset

Idea – move two vertical lines at distance s from left to right (*scanlines*)

Observation 1

- Suppose that the left scanline is at some position x_ℓ
- let X_ℓ denote the set of all trees with x coordinate in the interval $[x_\ell, x_\ell + s]$
- if there exist two trees in X_ℓ whose y coordinate differ by at least r, and no other tree in X_ℓ has y coordinate between these two – possible landing rectangle!

Idea – move two vertical lines at distance s from left to right (*scanlines*)

Observation 1

- Suppose that the left scanline is at some position x_ℓ
- let X_ℓ denote the set of all trees with x coordinate in the interval $[x_\ell, x_\ell + s]$
- if there exist two trees in X_ℓ whose y coordinate differ by at least r, and no other tree in X_ℓ has y coordinate between these two – possible landing rectangle!

Observation 2

- we cannot move scanlines continuosly ...
- what are the points of interest of either left or the right scanline?
  - hint – trees

For a given positions of scanlines, how to compute whether there exist two trees from the observation 1?

Exercise 3 - Forced Landing - Second approach

Observation 2

- we cannot move scanlines continuosly ...
- what are the points of interest of either left or the right scanline?
  - hint – trees

For a given positions of scanlines, how to compute whether there exist two trees from the observation 1?

Observation 2

- we cannot move scanlines continuosly ...
- what are the points of interest of either left or the right scanline?
  - hint – trees

For a given positions of scanlines, how to compute whether there exist two trees from the observation 1?

Observation 2

- we cannot move scanlines continuosly ...
- what are the points of interest of either left or the right scanline?
  - hint – trees

For a given positions of scanlines, how to compute whether there exist two trees from the observation 1?

Exercise 3 - Forced Landing - Second approach

Note

Filling out the missing details, and especially implementing it correctly, is very difficult! However, solving it is very rewarding!

Exercise 3 - Forced Landing - Summary

Tricks learned

- Prefix sums
- Scanline

General hint

- If a problem is modeled in k dimension, and it seems difficult, try first solving it if it has one less dimension and then extend the idea.
- *Note*: this is a hint, not a rule! (so it might not always work)

Exercise 3 - Forced Landing - Summary

Tricks learned

- Prefix sums
- Scanline

General hint

- If a problem is modeled in k dimension, and it seems difficult, try first solving it if it has one less dimension and then extend the idea.
- *Note*: this is a hint, not a rule! (so it might not always work)

Exercise 4 - Demolition

Easy solution – blow up floors $1, \ldots, \ell$

- blow up ℓ-th floor and simulate what happens after that
- complexity – $\mathcal{O}(n)$
- iterate over all floors – $\mathcal{O}(n^2)$ in total

Easy solution – blow up floors $1, \ldots, \ell$

- blow up ℓ-th floor and simulate what happens after that
- complexity – $\mathcal{O}(n)$
- iterate over all floors – $\mathcal{O}(n^2)$ in total

Easy solution – blow up floors $1, \ldots, \ell$

- blow up ℓ-th floor and simulate what happens after that
- complexity – $\mathcal{O}(n)$
- iterate over all floors – $\mathcal{O}(n^2)$ in total

Exercise 4 - Demolition

- S_ℓ – set of floors which you need to blow up additionally if the highest blown up floor is ℓ
- Relation of S_ℓ and $S_{\ell+1}$?
- Speed-up the simulation
  - using prefix sums – check in $\mathcal{O}(1)$ whether a floor i collapses if floors $i+1, \ldots, j$ collapse
  - for each floor in S_ℓ – check whether it survives the additional weight of the floor $\ell + 1$
  - however – still $\mathcal{O}(n^2)$ ... but enough for 50 points!

Exercise 4 - Demolition

- S_ℓ – set of floors which you need to blow up additionally if the highest blown up floor is ℓ
- Relation of S_ℓ and $S_{\ell+1}$?
- Speed-up the simulation
  - using prefix sums – check in $\mathcal{O}(1)$ whether a floor i collapses if floors $i+1, \ldots, j$ collapse
  - for each floor in S_ℓ – check whether it survives the additional weight of the floor $\ell + 1$
  - however – still $\mathcal{O}(n^2)$ ... but enough for 50 points!

Exercise 4 - Demolition

- S_ℓ – set of floors which you need to blow up additionally if the highest blown up floor is ℓ
- Relation of S_ℓ and $S_{\ell+1}$?
- Speed-up the simulation
  - using prefix sums – check in $\mathcal{O}(1)$ whether a floor i collapses if floors $i+1,\dots,j$ collapse
  - for each floor in S_ℓ – check whether it survives the additional weight of the floor $\ell+1$
  - however – still $\mathcal{O}(n^2)$ ... but enough for 50 points!

Exercise 4 - Demolition

Getting to $\mathcal{O}(n \log n)$

- consider some $f_1, f_2 \in S_\ell$
- define $A(f, \ell) := cap[f] - \sum_{i=1}^{\ell} w[i]$
- $A(f_1, \ell) > A(f_2, \ell)$
  - if f_2 doesn't collapse, neither will f_1
- simulation for the level $\ell + 1$ (hint)
  - get the element $f \in S_\ell$ with the smallest $A(f, \ell)$
  - check whether it collapses ...
- $A(f_1, \ell) > A(f_2, \ell) \rightarrow A(f_1, \ell') > A(f_2, \ell')$ for every $\ell' > \ell$

Getting to $\mathcal{O}(n \log n)$

- consider some $f_1, f_2 \in S_\ell$
- define $A(f, \ell) := cap[f] - \sum_{i=1}^{\ell} w[i]$
- $A(f_1, \ell) > A(f_2, \ell)$
  - if f_2 doesn't collapse, neither will f_1
- simulation for the level $\ell + 1$ (hint)
  - get the element $f \in S_\ell$ with the smallest $A(f, \ell)$
  - check whether it collapses ...
- $A(f_1, \ell) > A(f_2, \ell) \rightarrow A(f_1, \ell') > A(f_2, \ell')$ for every $\ell' > \ell$

General hints

- Don't rush to find the fastest solution
- Don't expect to come up with the brilliant idea just by looking at the problem – work towards it!
- Not everyone can solve every problem for 100 points; however, we will usually give a non-negligible number of points for slower solutions
  - if the problem requires $\mathcal{O}(n \log n)$ solution or faster, coding a $\mathcal{O}(n^2)$ one will very likely give you a certain number of points – which might save you on the exam!
  - not every slower solution will earn you some points – $\mathcal{O}(2^n)$ solution, for example, might be too slow if the optimal solution is $\mathcal{O}(n)$

General hints

- Don't rush to find the fastest solution
- Don't expect to come up with the brilliant idea just by looking at the problem – work towards it!
- Not everyone can solve every problem for 100 points; however, we will usually give a non-negligible number of points for slower solutions
  - if the problem requires $\mathcal{O}(n \log n)$ solution or faster, coding a $\mathcal{O}(n^2)$ one will very likely give you a certain number of points – which might save you on the exam!
  - not every slower solution will earn you some points – $\mathcal{O}(2^n)$ solution, for example, might be too slow if the optimal solution is $\mathcal{O}(n)$

General hints

- Don't rush to find the fastest solution
- Don't expect to come up with the brilliant idea just by looking at the problem – work towards it!
- Not everyone can solve every problem for 100 points; however, we will usually give a non-negligible number of points for slower solutions
    - if the problem requires $\mathcal{O}(n \log n)$ solution or faster, coding a $\mathcal{O}(n^2)$ one will very likely give you a certain number of points – which might save you on the exam!
    - not every slower solution will earn you some points – $\mathcal{O}(2^n)$ solution, for example, might be too slow if the optimal solution is $\mathcal{O}(n)$

General hints

- Don't rush to find the fastest solution
- Don't expect to come up with the brilliant idea just by looking at the problem – work towards it!
- Not everyone can solve every problem for 100 points; however, we will usually give a non-negligible number of points for slower solutions
  - if the problem requires $\mathcal{O}(n \log n)$ solution or faster, coding a $\mathcal{O}(n^2)$ one will very likely give you a certain number of points – which might save you on the exam!
  - not every slower solution will earn you some points – $\mathcal{O}(2^n)$ solution, for example, might be too slow if the optimal solution is $\mathcal{O}(n)$

Submitting

More fun interactively.

Remember

- Test locally!
- Times are deterministic. Submitting several times is a waste of resources

Judge Results

correct	you win!
wrong-answer	fix it, make testsets
timelimit	taking too long
compiler-error	fix it, compile locally
segmentation-fault	SIGSEGV: memory screwup
assertion-failure	SIGABRT: memory screwup or `assert()` or uncaught exception
run-error	nonzero exit status
forbidden	bad syscall or other safety limit
presentation-error	whitespace differences

Judge Results

correct	you win!
wrong-answer	fix it, make testsets
timelimit	taking too long
compiler-error	fix it, compile locally
segmentation-fault	SIGSEGV: memory screwup
assertion-failure	SIGABRT: memory screwup or `assert()` or uncaught exception
run-error	nonzero exit status
forbidden	bad syscall or other safety limit
presentation-error	whitespace differences

Judge Results

correct	you win!
wrong-answer	fix it, make testsets
timelimit	rethink approach
compiler-error	fix it, compile locally
segmentation-fault	SIGSEGV: memory screwup
assertion-failure	SIGABRT: memory screwup or `assert()` or uncaught exception
run-error	nonzero exit status
forbidden	bad syscall or other safety limit
presentation-error	whitespace differences

Judge Results

correct	you win!
wrong-answer	fix it, make testsets
timelimit	rethink approach
compiler-error	fix it, compile locally
segmentation-fault	SIGSEGV: memory screwup
assertion-failure	SIGABRT: memory screwup or `assert()` or uncaught exception
run-error	nonzero exit status
forbidden	bad syscall or other safety limit
presentation-error	whitespace differences

Judge Results

correct	you win!
wrong-answer	fix it, make testsets
timelimit	rethink approach
compiler-error	fix it, compile locally
segmentation-fault	use valgrind
assertion-failure	SIGABRT: memory screwup or `assert()` or uncaught exception
run-error	nonzero exit status
forbidden	bad syscall or other safety limit
presentation-error	whitespace differences

Judge Results

correct	you win!
wrong-answer	fix it, make testsets
timelimit	rethink approach
compiler-error	fix it, compile locally
segmentation-fault	use valgrind
assertion-failure	use gdb or valgrind
run-error	nonzero exit status
forbidden	bad syscall or other safety limit
presentation-error	whitespace differences

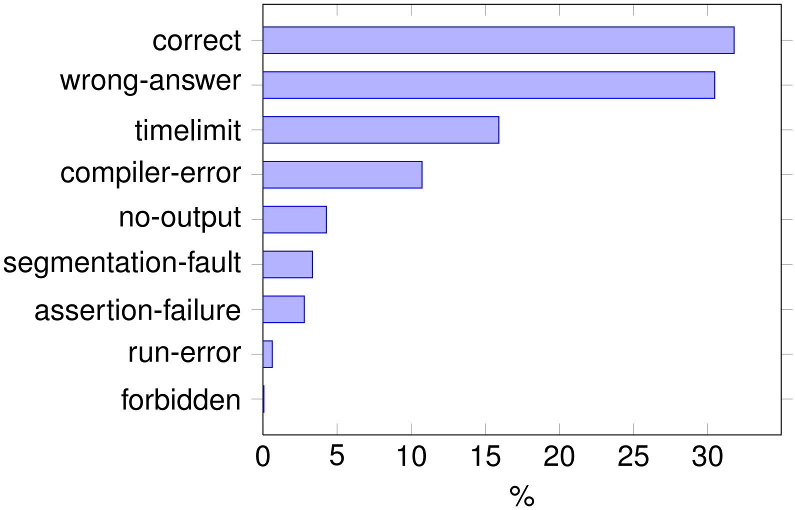
Judge Results

correct	you win!
wrong-answer	fix it, make testsets
timelimit	rethink approach
compiler-error	fix it, compile locally
segmentation-fault	use valgrind
assertion-failure	use gdb or valgrind
run-error	nonzero exit status
forbidden	bad syscall or other safety limit
presentation-error	whitespace differences

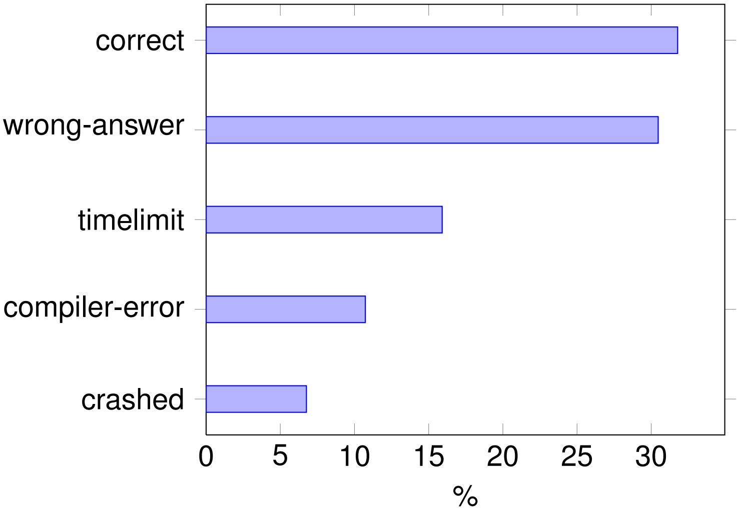
Judge Results

correct	you win!
wrong-answer	fix it, make testsets
timelimit	rethink approach
compiler-error	fix it, compile locally
segmentation-fault	use valgrind
assertion-failure	use gdb or valgrind
run-error	nonzero exit status
forbidden	stop it
presentation-error	whitespace differences

Distribution of results



Distribution of results



Debugging utilities discussed

valgrind

- + easy to use
- + verbose
- − occasionally hard to interpret
- − no interactive investigation
- − issues with rounding in CGAL

gdb

- + interactive
- + many possibilities
- − harder to use
- − post-segfault is often too late

debugging prints

GDB cheatsheet

`gdb` *prog*	start `gdb` on the *prog*
`run` *args*	run program with *args* as in shell
`bt`	print stack trace (backtrace)
`bt -`*N*	— of outermost *N* frames
`up`	up one frame (towards `main`)
`down`	down one frame
`print` *expr*	print the value of *expr*
`break` *func*	add breakpoint when entering *func*
`break` *line*	add breakpoint before *line*
`continue`	continue execution
`next`	— until next line in this function
`step`	— until next line anywhere
`quit`	quit

Take-aways of interactive session

01 Use assertions to catch input formatting mistakes (useful with your own testsets!)

02 Use `-Wall` at all times

03 `switch()` is finicky

04 Use `valgrind`
Beware of uninitialized values

05 Beware of array overruns; try `.at(i)`

06 Pointers invalidated across vector resizing

07 Be clear on static/heap/automatic data model

08 Stack overflow is also a segfault

09 Violating contracts can break STL algos