Practical Course: Algorithms for Programming Contests

TU München, winter term 2014/2015

Scripted by Daniel Hugenroth,
Maximilian Haslbeck,
Maximilian Hild.
mailto: maxf.hild@tum.de

9. Februar 2015

Inhaltsverzeichnis

1 First lecture (09/10/2014): Minimum spanning trees

Graph

Graph $= (V, E)$ where V denotes the nodes/vertices, E the edges
undirected: $E \subseteq \{\{a, b\} | a, b \in V\}$
directed: $E \subseteq \{(a, b) | a, b \in V\}$

Graph representations

- adjacency matrix:

	1	2	3	4
1	0	0	0	0
2	1	0	1	1
3	0	0	0	0
4	1	0	0	0

  disadvantages:
  - $|V|^2$ memory
  - find all adjacent nodes: go through column and row
  - advantage: lookup in $\mathcal{O}(1)$

- adjacency list:

1	2	3	4
↓	↓	↓	↓
	1		1
	3		
	4		

1	2	3	4
↓	↓	↓	↓
	$(1, 5)$		$(1, 5)$
	$(3, 6)$		
	$(4, -1)$		

Weights: $c : E \to \mathbb{R}$, $c(S) = \sum_{e \in S} c(e)$

Spanning tree

spanning tree: $G' = (V, S), S \subseteq E$, s.t. G' is a tree (connected, no cycles)
minimum spanning tree: spanning tree with minimum weight

Hints

- adding a constant weight to all edges in a graph will result in the same MST
- maximum spanning tree: multiply all weights by -1 and find MST

Lemma Let $E' \subseteq E$ s.t. there is a MST containing all edges of E'. Let T' be one of the subtrees induced by E' and let e be an edge having minimal weight connecting a node of T' with a node of $G \backslash T'$. Then, there is a MST of G containing $E' \cup \{e\}$.

1.1 Algorithm of Prim

Algorithm 1 Algorithm of Prim

1: **choose** a starting node $s \in V$
2: $visited[s] = true$
3: $visited[v] = false$ **for all** other $v \in V \backslash \{s\}$
4: **add** all neighbours of s to priority queue PQ ▷ runtime $\mathcal{O}(n+m)$
5: **keys**: weight of connecting edges ▷ $+n$ insert $\mathcal{O}(n)$
6: **while** not all nodes are visited **do** ▷ $+m$ decreaseKey $\mathcal{O}(m)$
7: $w = PQ.deleteMin()$ ▷ $+n$ deleteMin $\mathcal{O}(n \log m)$
8: **add** edge connecting to w to MST
9: $visited[w] = true$
10: **add** all unvisited neighbours of w to PQ
11: **if** already contained: decrease key
12: **end while** ▷ $\sum = \mathcal{O}(m + n \log m)$

running time ($|V| = n$, $|E| = m$)
everything without PQ: $\mathcal{O}(n+m)$
n insert: $\mathcal{O}(n)$
m decreaseKey: $\mathcal{O}(m)$
n deleteMin: $\mathcal{O}(n \log n)$

Hints

- In Java or C++ standard implementations of a priority queue there is no decrease key operation. In this case, just remove item from the priority queue and re-insert with lower key.
- Good implementations use a Fibonacci heap with insert, decrease key in $\mathcal{O}(1)$.
- If the exercise doesn't state different, always expect all general cases (like loops, multiple edges between two nodes, weight 0, ...)

1.2 Algorithm of Kruskal

Algorithm 2 Algorithm of Kruskal

sort all edges by increasing weight ▷ sort $\mathcal{O}(m \log m)$
initialize union find data structure, each node in an own component/set
for all edges e (sorted) **do**
  if nodes incident to e not in the same component **then**
    add e to MST ▷ $2m$ find $\mathcal{O}\big(m \cdot \alpha(m,n)\big)$
    merge components of nodes incident to e ▷ n union $\mathcal{O}\big(n \cdot \alpha(m,n)\big)$
  end if
end for ▷ $\sum = \mathcal{O}(m \log m)$

α is inverse Ackermann function which is a little slower than linear.

For the exam it is important to know the running times.

Course page: http://wwwmayr.in.tum.de/lehre/2014WS/conpra/
User: conpra, password: lea4TW!

Hints

- Start early
- Write your own test cases (beyond the samples)
- Ask questions if you have any problems (come by our office or write a clarification)

2 Second lecture (16/10/2014): Shortest paths

- Directed graph $G = (V, E)$
- weights $c : E \to \mathbb{R}$
- path from u to v:

$$(u, w_1), (w_1, w_2), \ldots, (w_{s-1}, w_s), (w_s, v) \in E, u, v, w_i \in V$$

- length: sum of the edges' weights
- problem: find shortest path between nodes
- all shortest paths from s to some other node form a tree rooted at s
- save predecessor (and distance) for each node
- all sources, single target: just switch source and target

Algorithm 3 Algorithm of Dijkstra

```
1: choose a starting node s ∈ V
2: visited[s] = true, pred[s] = null, dist[s] = 0
3: visited[v] = false, pred[v] = null, dist[v] = ∞ ∀v ∈ V\{s}
4: add all nodes adjacent to s to PQ, key: (dist[s] = 0+) weight of connecting edge
5: while not all nodes are visited do
6:     w = PQ.deleteMin()
7:     dist[w] = key in PQ
8:     add edge to the shortest path tree set pred[w] to other node on edge to w
9:     add all unvisited neighbours of w to the PQ, key: dist[w]+ edge weight
10:    if already contained then
11:        decreaseKey
12:    end if
13:    visited[w] = true
14: end while
```

Running time: like Prim's algorithm $\mathcal{O}(m + n \log n)$

Hint just use an array instead of the priority queue and iterate through all the array for deleteMin. This increases logarithmic running time to linear, but in practice it will not be significantly slower on everything that fits on a personal computer's memory.

Dijkstra is not usable if any node is < 0. It works with the logic that "if you add an edge to a path, it gets longer". Just imagine negative cycles.

The following algorithm will be able to handle negative edges.

Again, *dist* will contain shortest paths, here with at most i edges. After n steps, all shortest paths are found.

Algorithm 4 Algorithm of Bellman-Ford

```
1: choose a starting node s ∈ V
2: dist[s] = 0, pred[s] = null
3: dist[v] = ∞, pred[s] = null ∀v ∈ V\{s}
4: add s to queue q
5: while queue is not empty do
6:     w = queue.pop()
7:     for all neighbours v of w do
8:         if dist[v] > dist[w] + c((w, v)) then
9:             dist[v] = dist[w] + c((w, v))
10:            pred[v] = w
11:            add v to the queue
12:        end if
13:    end for
14: end while
```

negative cycles ⇒ infinite loop

When to stop?

- processing the queue one time = phase
- after i phases all shortest paths using at most i edges are found
  ⇒ at most m phases
  ⇒ at most n^2 loops

Running time

per phase: $\mathcal{O}(n+m)$ (dense graph $m = n^2 \Rightarrow \mathcal{O}(n^2)$)
total: $\mathcal{O}(n(n+m)) = \mathcal{O}(n \cdot m)$

Find negative cycles iterate from s through predecessors. If s shows up again, there's a negative cycle.

Hint in road network, only visit a node if it costs at most 30 minutes: add edge with -30 weight to this edge and use Bellman-Ford instead of Dijkstra.

How to find **longest paths**? Multiply the weights by -1 and use Bellman-Ford.

How to solve problems where **passing a node also takes time/cost**? Explode the node into two nodes connected by one edge with that nodes weight.

How to get **shortest paths for all pairs** in a graph? Use Floyd-Warshall in $\mathcal{O}(n^3)$.

3 Third lecture (23/10/2014): Flow

- directed Graph $G = (V, E)$
  capacity c: $E \to \mathbb{R}^+$
  source $s \in V$, sink $t \in V$
- flow f: $E \to \mathbb{R}$ with
  - capacity constraints:
    $0 \leq f(e) \leq c(e)\ \forall e \in E$
  - flow conservation: $\forall v \in V \backslash \{s, t\}$
    excess $e(v) = \sum_{e=(u,v)\in E} f(e) - \sum_{e=(v,u)\in E} f(e) = 0$ (amount of flow into node minus amount of flow out of node)
    $e(s) \geq 0$, $e(t) \geq 0$
- value of a flow: $v(f) := e(t) = -e(s)$

Example: [TODO]

Definition A **maximum flow** is a flow (satisfying the flow constraints) with maximum flow value.

Derivative problems:

1. consider a minflow → that's dumb! all zero (in contrast to longest/shortest path, the dual here is of no interest)
2. consider multiple sources → create a new supersource ss and connect it to all sources with infinite capacity edges, use ss as the source in your standard flow algorithm (similar with target).
3. limited source → consider there is a source that only can supply finitely many flow, i.e. $e(s) \geq -limit.$ to solve that, generate new supersource ss, connect it to the source with limited capacity edge.
4. capacity constraints on vertex → consider that some vertex can only handle certain limit capacity $c(v)$ (e.g. a router can only handle x packets per second). To do that, in a vertex v, split the vertex into two: v_{in} and v_{out}, connect the incomming arcs of v to v_{in} and the outgoing of v from v_{out}, then add an arc from v_{in} to v_{out} having a capacity $c(v)$.

3.1 How to solve it?

3.1.1 Residual Network

Definition of a Residual Network

- The residual Network of a graph $G = (V, E)$ is a directed graph $G' = (V, E')$, with capacities $c' \colon E' \to \mathbb{R}^+$
- forall $e = (u, v) \in E$
  1. if $f(e) < c(e)$ then add e to E' and set $c'(e) = c(e) - f(e)$
  2. if $f(e) > 0$ then add $e' = (v, u)$ to E' and set $c'(e') = f(e)$
- value of the flow can be increased along a path in the residual network, amount of additional flow: minimum capacity of this path in G'

Algorithm 5 Algorithm of Dinic

```
compute the residual network G' of G
while there is a s-t-path in G' do
    find a shortest path (unit distances)
    increase flow along this path
    update residual network G'
end while
```

Remarks

- works always
- running time: $\mathcal{O}(|V|^2 \cdot |E|)$
- use BFS (which Dijkstra does for unit lengths)
- fast to implement (really??)

3.2 Algorithm of Goldberg and Tarjan / Push-Relabel

Definition A **Preflow** is a flow $f : E \to \mathbb{R}_0^+$ with,

1. capacity constraint: $\forall e \in E. 0 \leq f(e) \leq c(e)$
2. flow conservation: $\forall v \in V \backslash \{s, t\}. e(v) := \sum_{e=(u,v) \in E} f(e) - \sum_{e=(v,w) \in E} f(e) \geq 0$

Remark: only difference to definition of flow is the $\geq$ in the flow conservation constraint!

- height: d: $V \to \mathbb{N}_0$
- legal edge (v, w): $d(v) = d(w) + 1$
- active node v: $e(v) > 0$

Algorithm 6 Algorithm of Goldberg and Tarjan

remove all nodes x that are either not reachable from s or do not reach t
initialize some Preflow: $\forall e = (s, v) \in E. f(e) := c(e)$, else $f(e) = 0$
for all $v \in V$ **do**
  if $e = (s, v) \in E$ **then**
    $f(e) = c(e)$
  else
    $f(e) = 0$
  end if
end for
initialize queue Q of active nodes
initialize distance function as the shortest distance (unit distance via BFS starting from t) to t without using s. ($d(s) := |V|$)
while Q not empty **do**
  pop first element v
  apply push(v) (see further down)
end while

Algorithm 7 push(v) procedure

while $e(v) > 0$ and there is a legal edge $e = (v, w)$ from v in the residual graph **do**
  push flow $\Delta = \min(e(v), c'(e))$ from v to w
  adjust excess values e(v), e(w) and residual graph.
  if $w \neq s$, $w \neq t$ and w not in Q **then**
    add w to Q
  end if
end while
if $e(v) > 0$ **then**
  relabel(v) (see further down)
end if

Remarks

- it is necessary to erase all nodes that are not reachable first, otherwise there are some nasty cornercases!
- nice running time $\mathcal{O}(|V|^3)$

Algorithm 8 relabel(v) procedure

set $d(v) = \min\{d(w)|(v,w) \in E'\} + 1$

note by maxH: example of the push/relabel algorithm at work:
`http://wwwmayr.in.tum.de/lehre/2013WS/ea/split/sub-Generic-Push-Relabel.pdf` (slide 498)

4 Seventh lecture (20/11/2014): Dynamic Programming

How to handle the big search space?

- "divide-and-conquer": split problems into smaller parts
- overlapping structure to solve subproblems without computing the same things again
- example: Dijkstra's algorithm
  - problem: shortest path from s to t
  - subproblem: shortest path from s to v that only uses nodes that are not further away
  - datastructure: array (dist, from)
- hash intermediate results
- create a hash table, store all results there
- before computing, look in the hash table if the result exists already
- exapmle: Fibonacci sequence

$$F_0 = 0 \qquad F_1 = 1 \qquad \dots \qquad F_n = F_{n-1} + F_{n-2}, \quad n \geq 2$$

4.1 Knapsack problem

- bag of size m
- list of n items with sizes a_i and score b_i
- *goal*: find a subset of items that fit into the bag with the maximum score
- 2^n possible subsets $\Rightarrow$ exponential search space (brute force too slow)
- subproblem $S_{i,j}$: use only first i items and only j space $(0 \leq i \leq n,\ 0 \leq j \leq m)$
- result to original problem: $S_{n,m}$

example

$n = 4$

sizes:	2	2	3	5
scores:	1	3	3	3

$n \backslash m$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1
2	0	0	3				
3	0	0	⋮				
4	0	0					X

$$S_{i,j} = \begin{cases} \max\left(S_{i-1,j-a_i} + b_i,\ S_{i-1,j}\right) & \text{if } j \geq a_i \\ S_{i-1,j} & \text{else} \end{cases}$$

running time: $\mathcal{O}(n \cdot m)$ $\leftarrow$ pseudo-polynomial, because polynomial in values of n and m, but exponential in the size of n and m ($\rightarrow$ log of n and m) (because we want to have the running time depending on the size of the input).

space: $\mathcal{O}(n \cdot m)$

- save only row of the array $\rightarrow \mathcal{O}(m)$

other examples:

- All pairs shortest paths
  Floyd-Warshall-Algorithms
- Edit distance
- Longest Common Subsequence
- Longest Increasing Subsequence

5 Eighth lecture (27/11/2014): Approximation

- NP-hard problems, many applicable in real life
- approximation vs. heuristics
- variation in approximability:
  - bin packing can be approximated within any factor > 1 (PTAS) ($1 + \varepsilon$ of the optimal solution, running time depending on ε)
  - maximum clique problem cannot be approximated
  - max. set, independent set, coloring have optimal approximation algorithms (every better approximation algorithm has the same running time as the naive solution)
- several techniques to approximate
  - greedy, e.g. coloring

    $G = (V, E)$, find minimal number of colors needed to color nodes

  - maximum cut

    $G = (V, E),\ c\colon E \to \mathbb{R}$, find cut $E\prime \subseteq E$ s.t. value of $E\prime$ is maximal

  - Knapsack (with unlimited amount of each item)

    $\frac{1}{2}$-approximation:

    · sort items decreasingly by ratio $\frac{v_i}{a_i}$ $\begin{smallmatrix}\leftarrow\text{value}\\ \leftarrow\text{weight}\end{smallmatrix}$

    · insert as many items as possible

    (X) [TODO: example]
  - local search

    · iteratively move from solution to solution

    · local changes until opt. is found or "time is up"

    · define neighbourhood relation (usually one component of solution is different)

    · exchanging 2 components: 2-opt

    (exchanging k components: k-opt)

    · how to choose next neighbour?

→ optimize some criterion (or minimize respectively): hill climbing

you might get stuck in local optima ...

- Simulated Annealing (from metallurgy)
  - temperature determines the acceptance probability of a solution
  - set temperature high initially, then lower it gradually
  - while(break condition)

    apply k-opt step to get x'

    accept if $f(x, x', T) > R$

    e.g. $e^{-\frac{\text{diff}}{T}} > R$

    after some time, lower T

    The choice of the parameters is important!

    → initial temperature: quite high (initial acceptance rate is between 40% and 60%

    → $f(x, x', T) \xrightarrow{T \to 0} 0$

    → cooling scheme:

    $$T = \alpha T \qquad \text{or} \qquad T = \frac{T}{1 + \beta T}$$

  - conditions on lowering T:

    → number of steps

    → number of consecutive improving steps depend on the input (size)
  - termination:

    → number of steps

    → number of non-improving steps
- minimum vertex cover
- TSP (travelling salesperson problem)
- SAT

Remarks

- as we are talking about randomization, it might be possible that you upload the same solution twice and get two different results.

- the time limit is not pretty strict this week, so just run it like 20 times and output the best solution
- you won't have to find the optimal solution, but e.g. 50% or 10% of the solution

6 Ninth lecture (11/12/2014): Online Algorithms

offline:

- whole input is given initially
- all decisions can be made with knowledge of the whole input

online:

- does not have input initially
- input arrives incrementally over time
- makes decisions without knowledge of any future input

example:

- selection sort (offline)
- insertion sort (online)

formal model: alg A

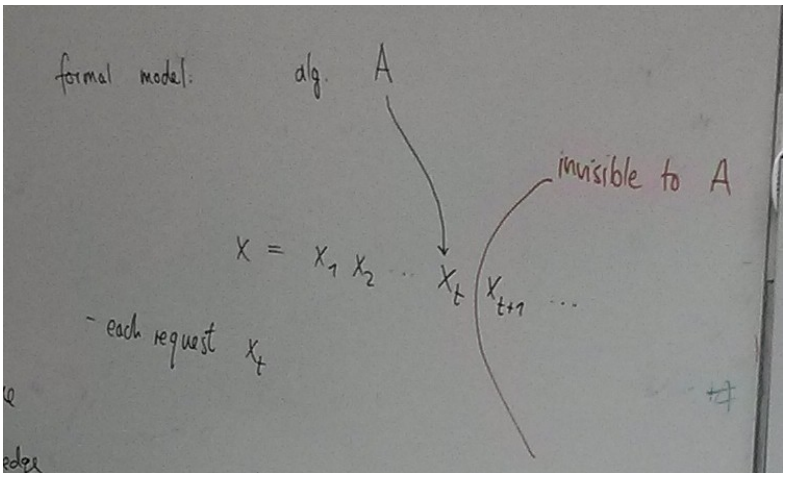


Abbildung 1: Model: offline algorithm

- each request x_t has to be served without knowing x_{t+1}
- comparison to optimal offline algorithm
  - KNOWS the whole input in advance
  - can serve requests optimally with optimized cost
- worst-case analysis
  - over all distributions of x

- generally no assumptions about this distribution

An online algorithm A is *c-competitive*:

$$\exists a > 0 : \ \forall x : A(x) \leq c \cdot OPT(x) + a$$

6.1 Ski rental problem

- whole class of problem
- choice between
  - continuing to pay the repeating cost or
  - paying a one-time cost eliminating the repeating cost
- buy/rent

→ going skiing for an unknown number of days

$$1 \text{ €} \text{ / day} \qquad\qquad 10 \text{ €} \text{ to buy}$$

decide every day whether to rent or buy

offline algorithm

$$< 10 \text{ days} : \ \text{rent every time}$$
$$\geq 10 \text{ days} : \ \text{buy for } 10 \text{ €}$$

break-even-algorithm (best deterministic alg.)

rent for 9 days
buy on 10^{th} day

$$< 10 \text{ days} : \qquad \Rightarrow \text{optimal}$$
$$> 10 \text{ days} : \qquad 9 \text{ €} + 10 \text{ €} = 19 \text{ €}$$

randomization: 1.9-competitive

flip a coin $\begin{cases} \nearrow \text{heads: buy on day 8} \\ \searrow \text{tails: buy on day 10} \end{cases}$

$$\geq 10 \text{ days: pay } \frac{1}{2}(7+10) + \frac{1}{2}(9+10)€ = 18€$$

best randomized alg.:

$$\frac{e}{e-1}\text{-competitive} \approx 1.58\ldots$$

6.2 Online scheduling

6.2.1 1

- set of n tasks with different execution times
- set of m machines
- goal: minimize the completion time of the last task

  $\rightarrow$ greedy alg.: schedule task on the machine with the smallest load

  (ist scheduling)

  $$(2 - \frac{1}{m})\text{-competitive}$$

6.2.2 2

- set of n tasks with different execution times and release dates
- 1 machine
- goal: minimize the sum of all completion times
- idea: schedule short tasks earlier and long tasks later
- greedily scheduling the shortest available task might be bad

Hints

- if you get a time limit this week, there are two ways:
  - either your solution is really too slow,
  - or you ran into a deadlock or something, like you are waiting for some input that isnt coming when instead you should print something
- create local test cases

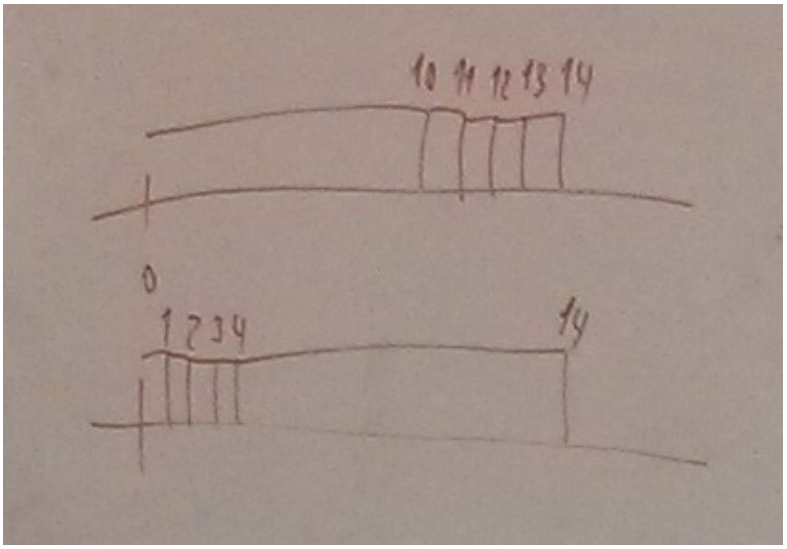


Abbildung 2: Model: scheduling

- input exactly like in the problem states

7 Tenth lecture (18/12/2014): Number theory

"Basically the study of integers where you restrict all calculations to just the set of integers, so you wont look at real numbers, complex numbers or even more complicated things. Most of the times its just integers. " (Chris)

- study of integers
- since around 1800 BC
    - Pythagorean triples in Mesopotamia
- many ideas are easy to understand
- old/famous

```
500-200 BC                300-500 CE     12th c      17th-18th ce
classical Greece          China, India
------------------------------------------------------------------------>
Pythagoras, Plato,        Sun Tzu        Fibonacci   Fermat, Euler, Gauss
Euclid
```

7.1 Subdivisions

- Elementary Tools
- Analytic Number Theory

  tools/methods from real/complex analysis
- Algebraic Number Theory

  tools/methods from Algebra
- Diophantine Geometry

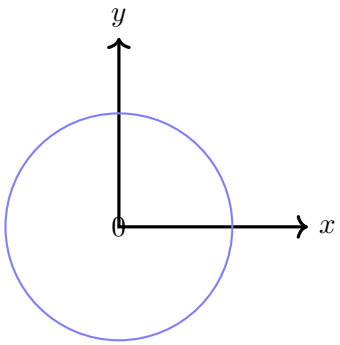
  polynomial equation with integral solutions

  $$x^2 + y^2 = 1$$

  "points on curves"

7.2 Sieve Theory

- techniques to count/estimate size of a sifted subset
- approximate sifted set by simpler superset (primes by k-almost primes)
- Sieve of Eratosthenes (3^{rd} century BC)



- prime sieve
- iteratively mark composites

  multiples of 2, 3, ... $2, 3, \ldots$

- $$\begin{pmatrix} \not{1} & \textcircled{2} & \textcircled{3} & \not{4} & \textcircled{5} & \not{6} \\ \textcircled{7} & \not{8} & \not{9} & \not{10} & \textcircled{11} & \not{12} \\ \textcircled{13} & \not{14} & \not{15} & \not{16} & \textcircled{17} & \not{18} \\ \textcircled{19} & \not{20} & \not{21} & \not{22} & \textcircled{23} & \not{24} \end{pmatrix}$$
- faster than trial division
- efficient for smaller primes

  poly-time algorithms since 2002

7.3 Euclidean algorithm

- computes $gcd(a, b)$ greatest common divisor
- can be generalized to polynomials etc.
- a and b coprime: $gcd(a, b) = 1$
- $lcm(a, b)$ lowest common multiple
- compute Euclidean divisors

$$\begin{aligned} q_1, \ldots, q_k, & \qquad r_0, \ldots, r_{k+1} \\ r_0 = a & \qquad r_1 = b \\ r_{i+1} = r_{i-1} - q_i r_i & \qquad \text{until } r_{k+1} = 0 \end{aligned}$$

7.3.1 Extended Euclidean algorithm

- $gcd(a, b) = ax + by$
- use quotients!
- $s_0, \ldots, s_{k+1}, t_0, \ldots, t_{k+1}$ Bézout coefficients
  $s_0 = 1 \qquad t_0 = 0$
  $s_1 = 0 \qquad t_1 = 1$
  $s_{i+1} = s_{i-1} - q_i s_i \; t_{i+1} = t_{i-1} - q_i t_i \qquad t_0 = 0$
  $\rightarrow r_k = gcd(a, b) = a \cdot s_k + b \cdot t_k$

Chinese Reindeer Theorem ("Merry christmas!")

"It's not that he got really old, its only they don't know exactly when he lived. " (Chris)

- Sun Tzu: 3^{rd}–5^{th} c
- completed and proven later
- determine x that when divided by given divisors n_i leaves given remainders a_i
- $n_1, \cdots, n_k$ pairwise coprime

$$\Rightarrow \exists x\colon \; x \equiv a_i \mod n_i \;\; \forall i$$

and all solutions are congruent $\mod N = n_1 \cdots n_k$

$$x \equiv 2 \mod 3$$

$$x \equiv 3 \mod 4$$

$$x \equiv 1 \mod 5$$

- bruteforce:
  - convert congruences into sets

$$\{2, 5, 8, 11, 14, \ldots, 68, 71, 74, \ldots\}$$

$$\{3, 7, 11, 15, \ldots, 67, 71, 75, \ldots\}$$

$$\{1, 6, 11, 16, \ldots, 66, 71, 76, \ldots\}$$

  - intersect

$$\{11, 71, 131, \ldots\}$$

- algebraic:

$$x = 2 + 3t$$

$$x = 3 + 4s$$

$$x = 1 + 5u$$

$$\rightarrow x = 2 + 3t = 3 + 4s \equiv 3 \mod 4$$

$$\Rightarrow 3t \equiv 1 \mod 4$$

$$t \equiv 3 \mod 4$$

$$t = 3 + 4s$$

$$\rightarrow x = 2 + 3t = 2 + 3(3 + 4s) = 11 + 12s \equiv 1 \mod 5$$

$$\Rightarrow s = 0 + 5u$$

$$x = 11 + 12s = 11 + 12(0 + 5u) = 11 + 60u$$

- constructive: use Ext. Euclid: $r_i n_i + \underbrace{s_i \frac{N}{n_i}}_{=:e_i} = 1$

$$r_i n_i + e_i = 1 \qquad N = 3 \cdot 4 \cdot 5 = 60$$

$$-13 \cdot 3 + 2 \cdot 20 = 1 \qquad e_1 = 40$$

$$-11 \cdot 4 + 3 \cdot 15 = 1 \qquad e_2 = 45$$

$$5 \cdot 5 + (-2) \cdot 12 = 1 \qquad e_3 = -24$$

$$\uparrow e_i \cdot s$$

$$\Rightarrow e_i = \begin{cases} 1 \mod n_i \\ 0 \mod n_j, j \neq i \end{cases}$$

$$x = \sum_{i=1}^{k} a_i e_i$$

$$x = 2 \cdot 40 + 3 \cdot 45 + 1 \cdot (-24)$$

$$191 = 11 + 3 \cdot 60$$

  - all other solutions are congruent to 191 $\mod 60$ equals 11 $\mod 60$
  - different implementations of Euclid may yield different results, but all will provide a correct solution in the end

8 Twelfth lecture (15/01/2015)

Next Week: Contest week

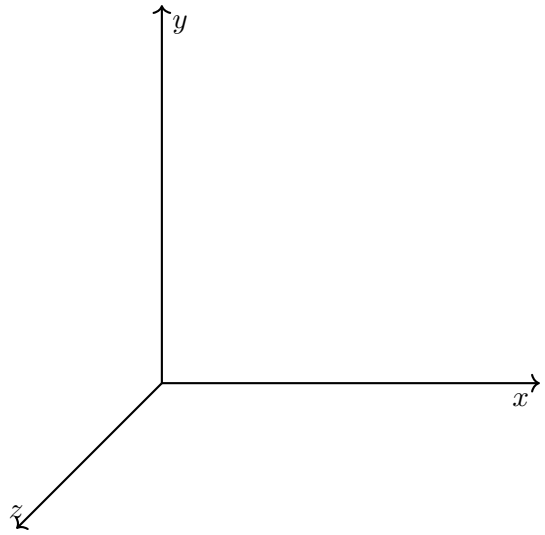
Send clarification with team member and team name, or stating you need a team mate.

Saturday, 24/01/2015 11 am, ICPC Winter Contest

8.1 Projective Geometry

8.1.1 Points

- use three-dimensional vectors to save two-dimensional points
- $(0, 0, 0)$ is not allowed, everything else is
- represent the point (a, b) as the vector $(a, b, 1)$ (homogenization)
- scalar multiples represent the same point
- normalization: set last coordinate to 1: $(a, b, c) \hat{=} (\frac{a}{c}, \frac{b}{c}, 1), c \neq 0$
- de-homogenization: normalize, remove last component



8.2 Lines

- use three-dimensional verctos too
- generate a plane by the line and $(0, 0, 0)$, take a normal vector

- scalar multiples represent the same line
- one line is not contained in $\mathbb{R}^2$: $(0, 0, 1)$

8.3 Now its getting interesting ...

Point p is on line l	$\Leftrightarrow$	$\langle p, l \rangle = 0$
Line through points p, q:		$p \times q$ (outer product)
Points p, q, r are collinear	$\Leftrightarrow$	$\langle p \times q, r \rangle = 0$
Intersection of two lines m, n:		$m \times n$ (outer product)
Intersection of the diagonals of a quadrilateral spanned by points a, b, c, d		$(a \times c) \times (b \times d) = (c \times a) \times (b \times d)$ $= (b \times d) \times (a \times c) = a \times c \times b \times d$

8.4 Infinite Points

- intersection of parallel lines
- example: $(1, 0, 1) \times (2, 0, 1) = (0, 1, 0)$
- interpretation: direction of the lines
- connection of infinite points: $(0, 1, 0) \times (1, 0, 0) = (0, 0, 1)$
- parallel line m to line l through point p
  $q = l \times (0, 0, 1)$
  $m = q \times p$
  $= l \times (0, 0, 1) \times p$
- orthogonal direction of $(a, b, 0)$ is $(b, -a, 0) = (a, b, 0)^\perp$
- projection of point p on line l
  $q = l \times (0, 0, 1)$
  $m = p \times q^\perp$
  $p' = m \times l$
  $\Rightarrow p' = p \times (l \times (0, 0, 1))^\perp \times l$
- area of a triangle through points p, q, r:

$$A = |\det(p, q, r)|$$

8.4.1 Projective Transformations

- Linear map on homogenous coordinates $\rightarrow$ matrix $\in \mathbb{R}^{3\times3}$
- scalar multiples correspond to the same map

- defined by the image of 4 non-collinear points

$$\begin{aligned} a \mapsto a', b \mapsto b', c \mapsto c', d \mapsto d' \\ \Rightarrow Ma = a', Mb = b', Mc = c', md = d' \end{aligned}$$

  $\Rightarrow$ solve system of equations $\rightarrow$ gives M

- it is possible to shift, mirror, scale and rotate

9 Thirteenth lecture (22/01/2015): Comparison of different I/O methods

- Next Week: Contest
- Teams of two students each
- Create teams? Write a clarification before you make your first submission!
- Team logins: team_teamcaptainslogin
- Invite your friends!
- 10 problems for 6 points each, we will only use the normal 28 points for the total number of points (everything else is bonus).
- asking for half of the points is working if the approach is right
- Winter Contest: Saturday, 24/01/2015, Rechnerhalle
- Compete with other teams from Europe!
- There will be pizza!
- ... and bonus points!
- 5 points for the first two problems solved, 2 points for each other problem solved

9.1 Input and output in C++ and Java

In most cases in this lecture, input speed was not important. But in the contest, there is often a lot of input/output which has to be read/written.

9.1.1 C++

Input:		
cin	4.7 s	5 mio. numbers
scanf	1.6	5 mio. numbers
$\Rightarrow$	factor	≈ 3
Output:		
cout	12 s	
printf	1.1 s	
$\Rightarrow$	factor	≈ 10

9.1.2 Java

(on different hardware, so don't compare with C++)

Input Scanner	3,4 s
BufferedReader	450 ms
Output:	
println	15 s
format	25 s
BufferedWriter	700 ms
StringBuilder with println	450 ms
StringBuilder with BW	450 ms

Interesting fact BufferedReader is about as fast as scanf, StringBuilder is about as fast as printf.

9.2 Floating Point Precision

handwritten: $5.3821 => 5.3821 \cdot 10^0$
$0.0012\_\_\_- > 1.2\_\_\_ \cdot 10^{-3}$

9.2.1 Multiplication

$1.23 \cdot 10^1 \times 2.34 \cdot 10^{-5} =? \cdot 10^{-4}$

9.2.2 Division

$1.23 \cdot 10^1 : (2.34 \cdot 10^{-5} =? \cdot 10^6$

9.3 Addition

  1.234
+ 0.00002345
= 1.23402345

9.4 Subtraction

  1.234
− 1.233
= 0.001???

9.5 Hints for the contest

- Look at the scoreboard!

- Problems solved in the first 10–15 mins HAVE to have an (easy and) short solution
- Teams behind you on the scoreboard are probably worse than you. If they solved some problem you didn't solve, you should have a look at that one.
- Last 30 minutes: do NOT start new problems that you think you can implement fast. That NEVER works. Istead, try to fix problems, that you already coded but didn't work.
- Make use of a language's advantages
- E.g. for DateTime and BigIntegers, Polygon intersections, line geometry ... use Java!!!
- If speed is the key, better use C++ (like bruteforce)
- recursion in C++ is of course very slow if you pass big arrays by value

10 Fourteenth lecture (29/01/2014): Contest Week Solutions

If you need any of the solutions, write me an e-mail or go to the lecturers' office.

The following is taken from last year

11 Lecture five: Brute Force Algorithms

Brute force, also known as *exhaustive search* or *generate and test* means to generate solutions and validate them.

+ simple

+ correct (if you go through all search space)

- inefficient

+ good when errors are critical

+ good when computational power/time is not an issue

- bad when input is too big

11.1 Linear search

sequentially finding divisors of n: go through all integers $1 \dots \sqrt{n}$

11.2 Pseudocode

Algorithm 9 Pseudo code for simulated annealing

```
x = first()
while n ≠ null do
    if valid(x) then
        output x
    end if
    x = next()
end while
```

11.3 Examples

- number of anagrams of a word
- number of permutations of n numbers: $n!$
- number of functions from $[n]$ to $[k]$: k^n
- binomial coefficient $\binom{n}{k}$, disregarding orders (number of poker hands)

→ estimate number of operations used by a brute force algorithm

password safety checker:

256 characters, length $\leq 5 \rightarrow 256^5 > 10^12$
256 characters, length $\leq 6 \rightarrow 256^6 > 2 \cdot 10^14$
256 characters, length $\leq 10 \rightarrow 256^10 > 10^25$

finding divisors:

a 32-bit number n has up to 10 decimal digits
$\rightarrow \sqrt{n}$ has up to 5 digits $\rightarrow 10^5$
a 64-bit number n has up to 20 decimal digits
$\rightarrow \sqrt{n}$ has up to 10 digits $\rightarrow 10^{10}$

queens problem

$\rightarrow 64^8 \approx 2.8 \cdot 10^{14}$ configurations (but allows queens on top of each other)
$\rightarrow \frac{64!}{56!} = 64 \cdot 63 \cdot 62 \cdot 61 \cdot 60 \cdot 59 \cdot 58 \cdot 57$ (but many solutions are the same as we cannot distinguish between queens)
$\rightarrow \binom{64}{8} \approx 4.4 \cdot 10^9$

$\Rightarrow$ thinking about the problem can reduce the search space drastically and make our brute force algorithm faster

find all integers in $\{1, \ldots, C\}$ divisible by x
naive algorithm: C tests
better: $\frac{C}{x}$ (only multiples of x)

another optimization: reorder solution space to start with promising solutions first

how to generate all permutations?

in lexicographic order:

start with smallest sequence	1234
find largest index k s.t. $a_k < a_{k+1}$	1243
if k doesnt exist, then this is the last one	1342
find largest index l s.t. $a_k < a_l$	1324
swap values a_k and a_l	$\vdots$
reverse sequence from a_{k+1} up to a_n	4321

12 Seventh lecture: Backtracking

- search space has tree structure
- build partial solutions (internal nodes)
  - check if it can be completed
    if not: diregard the whole subtree (pruning)
  - DFS, recursively, incrementally

However, this requires that the concept of a partial solution must exist.

12.1 Examples

Satisfiability: A partial solution would be assignments to some variables (e.g. the k first variables $V_1 ... V_k$ are set). In the k-th level of the tree, the first k Variables are set. At the root $k = 0$ (no variable is set), at the nodes a complete solution is present.

Other: Queens Problem, Coloring, Combinatorial Optimization (Knapsack, ...), Puzzles (Crossword, Sudoku, ...)

Constraint Satisfaction Problems (CSP): Find assignment to a set of variables s.t. some constraint is satisfied. CSP denotes a general class of problems that are solvable using backtracking.

12.2 Implementation

For general implementation of backtracking see algorithm **??**. We do now discuss the **CSP**, which can be simply plugged into the algorithm **??**.

The goal is to find integers $(x_1, ..., x_n) \in [m]^n$ which satisfy a constraint F (boolean function: $[m]^n \to \{true, false\}$).

A partial solution is a list of integer $(y_1, ..., y_j), k \leq n$. At the root $k = 0$

For the algorithm we plug in:
c.length = k;
next(c) = $\{(c_1, ..., c_k, 1), ..., (c_1, ..., c_k, m)\}$[1];
completed(c) = (c.length == n);
valid(c) = is F potentially satisfiable.

[1] However, the order of the enumeration does not matter

13 Eighth lecture: Geometry

Example: decide on which side of a line a point p is lying ("left" or "right")

- sides of a line from point a to b
- if $a.x = b.x$ (vertical line)

$$M_0 = \{p \in \mathbb{R}^2 | p.x < a.x\}$$
$$M_1 = \{p \in \mathbb{R}^2 | p.x = a.x\}$$
$$M_2 = \{p \in \mathbb{R}^2 | p.x > a.x\}$$

- else represent line as $y = mx + n$

$$m = \frac{a.y - b.y}{a.x - b.x}$$
$$n = a.y - m \cdot a.x$$

$$M_0 = \{p \in \mathbb{R}^2 | m \cdot p.x + n < p.y\}$$
$$M_1 = \{p \in \mathbb{R}^2 | m \cdot p.x + n = p.y\}$$
$$M_2 = \{p \in \mathbb{R}^2 | m \cdot p.x + n > p.y\}$$

- polygon: plane shape bounded by a finite closed chain of line segments
- simple polygon: edges do not intersect (despite at the corresponding vertices)
- convex set $M \in \mathbb{R}^2$: $\forall p, g \in M$ the line segment between p and q is contained in M
- convex hull of $M \subseteq \mathbb{R}^2$: the smallest convex set containing M
- one vertex of the convex hull: e.g. lexicographically smallest point:
  smallest x-coordinate, use y-coordinate as a tie-breaker $\rightarrow \mathcal{O}(n)$

13.1 Gift Wrapping Algorithm

a.k.a. Jarvis March

- find one vertex p of the convex hull, add it to the convex hull
- iterate through all other points: select a point q that is not a vertex of the convex hull yet, but all points are on the same side of pq
- if this is not possible return the convex hull
- add q to the convex hull, set $p = q$, restart

13.1.1 running time

$\mathcal{O}(n^2)$, n is the number of vertices

or $\mathcal{O}(n \cdot h)$, h is the size of the convex hull ($\rightarrow$ output-sensitive running time)

Algorithm 10 gift wrapping algorithm

```
procedure JARVISMARCH(c)
    if completed(c) then
        return true;
    end if
    if cannotBeCompleted(c) then
        return false;
    end if
    for s in next(c) do
        if valid(x) and backtrack(s) then
            return true
        end if
    end for
    return false
end procedure

start with: backtrack(root)
```

13.2 Graham's Scan

"I think you can guess the name of the inventor of this algorithm." (Stefan T.)

- find one vertex p of the convex hull, add it to the convex hull.
- order all points q by the angle between pq and the y-axis (counter-clockwise)
- set q to the first point, add q to the convex hull
- iterate through all other points r
    - if r is on the right side of pq
        - remove q from the convex hull
        - add r to the convex hull
        - $q = r$
    - else
        - add r to the convex hull

Abbildung 3: Mr Scanner, inventor of Graham's Scan algorithm

· $p = q$, $q = r$

13.2.1 running time

$\mathcal{O}(n \cdot \log n)$, optimal (comes by sorting, rest is even linear)

13.3 Kirkpatrick-Seidel Algorithm

$\mathcal{O}(n \log h)$, optimal output-sensitive

13.4 Chan's Algorithm

see linked course material, not needed in this course

13.5 Andrew's Monotone Chain Algorithm

like Graham's, but go half clock- and half counter clockwise

Algorithm 11 Graham's algorithm

```
procedure GRAHAMSALGORITHM
    find one vertex p of the convex hull, add it to the convex hull.
    order all points q by the angle between pq and the y-axis (counter-clockwise)
    set q to the first point, add q to the convex hull
    for iterate through all other points r do
        if r is on the right side of pq then
            remove q from the convex hull
            add r to the convex hull
            q = r, iterate
        else
            add r to the convex hull
            p = q, q = r
        end if
    end for
end procedure
```

13.6 Quick Hull

(named like quick sort)
connect two points of convex hull, find points on left and right with greatest distance to connection line, add them to convex hull and do the same with the new lines of convex hull

13.6.1 expected running time

$\mathcal{O}(n \log n)$
worst case: $\mathcal{O}(n^2)$

13.7 Point in Polygon

"If you know the convex hull, a problem is much easier. " (Stefan T.)

- give points $p_0, \ldots, p_{n-1}, q$
- test whether q is inside the polygon defined by $p_0, \ldots, p_{n-1}$
- given convex hull: check whether q is on the same side of $p_i p_{(i+1)\%n}$ as the other points for all $0 \leq i < n$

general case:

Abbildung 4: Mr Al Go Rithm

13.7.1 Ray Casting Algorithm

- choose a ray starting at q (e.g. a unit vector as direction)
- for each edge of the polygon check whether the edge intersects the ray
- count the number of intersections m
- m odd iff q is inside the polygon

running time: linear

be careful with floating point rounding etc.