Solution Outlines

Jury

GCPC 2013

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

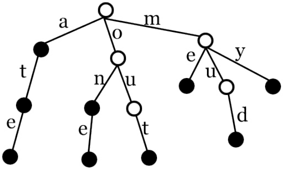
acm International Collegiate Programming Contest

Boggle

- 4×4 grid of characters
  $\Rightarrow$ enumerate all $\approx 300\,000$ words
- linear lookup in dictionary with $300\,000$ words is obviously too slow

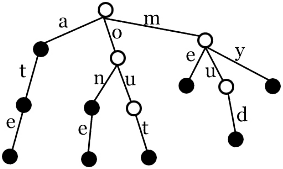
Boggle

- 4×4 grid of characters
  $\Rightarrow$ enumerate all $\approx 300\,000$ words
- linear lookup in dictionary with $300\,000$ words is obviously too slow
- compute trie with all dictionary words in linear time
- check if a word is possible while enumerating



Boggle

- 4×4 grid of characters
  $\Rightarrow$ enumerate all $\approx 300\,000$ words
- linear lookup in dictionary with 300 000 words is obviously too slow
- compute trie with all dictionary words in linear time
- check if a word is possible while enumerating
- second option: use binary search in sorted dictionary instead of trie (don't forget to prune then)



Booking

Assign bookings to rooms

- Conflict / compatibility graph (bookings are vertices, conflicts/compatibilities are edges)
- Coloring / clique partitioning

Booking

Assign bookings to rooms

- Conflict / compatibility graph (bookings are vertices, conflicts/compatibilities are edges)
- Coloring / clique partitioning
- Problem has special structure (timings)
- Equals register allocation problem for variables in data/control-flow graph
- Solution: Left-Edge Algorithm (runs in at most $\mathcal{O}(B^2)$ (worst case))

Booking

- Read in bookings
    - convert dates to time stamps (e.g. with Java DateFormatter)
    - don't forget that 2016 is a leap year
- Sort bookings by arrival date
- Assign same "color" to bookings that do not overlap (Left-Edge Algorithm)
- Don't forget the cleaning time

Booking

However, we don't care for the actual room assignments
$\Rightarrow$ forget about Left-Edge Algorithm

Booking

However, we don't care for the actual room assignments
$\Rightarrow$ forget about Left-Edge Algorithm

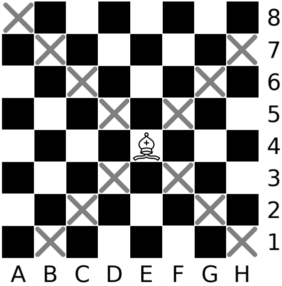
Alternative solution:

- Read in dates as before
- Add cleaning time to departures
- Store arrivals and departures individually (as "events") in one array
- Use a flag to indicate which events are arrivals and which ones are departures

Booking

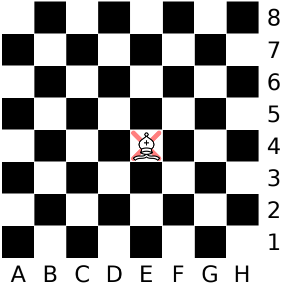
- Sort events (dates) by time stamp (if time stamps are equal, departures come before arrivals)
- Iterature through array and maintain a counter:
  - increment, if event (time stamp) marks an arrival
  - decrement, if event (time stamp) marks a departure
- Output maximum value of counter
- Runs in $\mathcal{O}(B \cdot \log B + B) = \mathcal{O}(B \cdot \log B)$

Chess



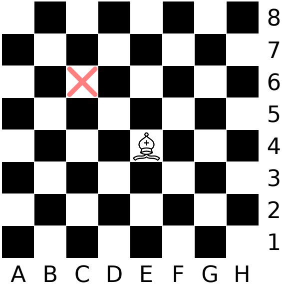
- Easiest problem of the set

Chess



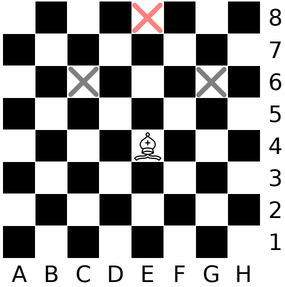
- Easiest problem of the set
- You are already at the goal

Chess



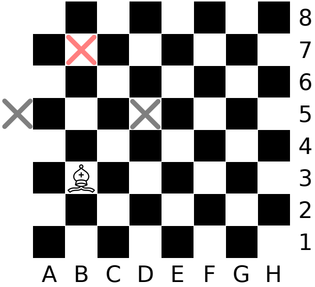
- Easiest problem of the set
- You are already at the goal
- You can move directly

Chess



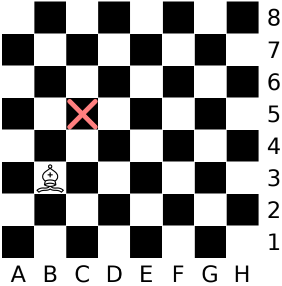
- Easiest problem of the set
- You are already at the goal
- You can move directly
- You can reach the goal in two moves
- Two possible ways

Chess



- Easiest problem of the set
- You are already at the goal
- You can move directly
- You can reach the goal in two moves
- Two possible ways
- One may be invalid

Chess



- Easiest problem of the set
- You are already at the goal
- You can move directly
- You can reach the goal in two moves
- Two possible ways
- One may be invalid
- You cannot reach the goal at all

Kastenlauf

- read coordinates of locations
- build graph;
  - location becomes node
  - insert edge if distance not greater than 1 000

Kastenlauf

- read coordinates of locations
- build graph;
  - location becomes node
  - insert edge if distance not greater than 1 000
- in this graph: is end node reachable from start node?
- DFS, or BFS, or anything...
- $\mathcal{O}(n^3)$ solutions acceptable, although better ones do exist.

No Trees but Flowers

- Volume computation of rotational body
- Integration of 1D function $f(x) = a \cdot e^{-x^2} + b \cdot \sqrt{x}$
- $V = \int_0^h f(x)^2 \cdot \pi$

No Trees but Flowers

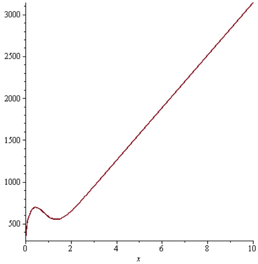
- Volume computation of rotational body
- Integration of 1D function $f(x) = a \cdot e^{-x^2} + b \cdot \sqrt{x}$
- $V = \int_0^h f(x)^2 \cdot \pi$
- e^{-x^2} cannot be analytically integrated
- Use numerical integration instead

No Trees but Flowers

- Volume computation of rotational body
- Integration of 1D function $f(x) = a \cdot e^{-x^2} + b \cdot \sqrt{x}$
- $V = \int_0^h f(x)^2 \cdot \pi$
- e^{-x^2} cannot be analytically integrated
- Use numerical integration instead
- Naive implementation usually too slow
- At least trapezoidal rule required

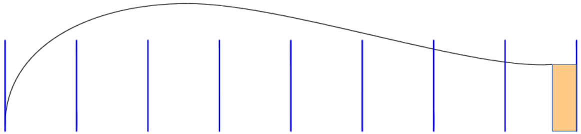
How to estimate the required mesh width?

- Absolute accuracy specified
- Maximum relative accuracy required for largest integral value
- $\rightarrow a = 10, b = 10, h = 10$
- Offline convergence test gives valid mesh width



Alignment of Discretization

- Upper bound (h) may not be an integer.
- Align discretization to integration bounds.



Peg Solitaire

- Estimate the number of possible game developments:

Peg Solitaire

- Estimate the number of possible game developments:
  - not more than 2 available moves/peg on average

Peg Solitaire

- Estimate the number of possible game developments:
  - not more than 2 available moves/peg on average
  - each move eliminates one peg

Peg Solitaire

- Estimate the number of possible game developments:
  - not more than 2 available moves/peg on average
  - each move eliminates one peg
  - $2^{P-1} \cdot P! = 5\,160\,960$
    for $P = 8$ pegs

Peg Solitaire

- Estimate the number of possible game developments:
  - not more than 2 available moves/peg on average
  - each move eliminates one peg
  - $2^{P-1} \cdot P! = 5\,160\,960$
    for $P = 8$ pegs

$\Rightarrow$ Small enough to use backtracking (without further improvements)

Ringworld

- Given n intervals on a circle, choose one node inside each interval so that no node is used twice.

Ringworld

- Given n intervals on a circle, choose one node inside each interval so that no node is used twice.
- First consider the problem on a line. Then a simple greedy algorithm works:
  - scan the nodes from left to right,
  - whenever we counter the right endpoint b_i of an interval $[a_i, b_i]$, choose the leftmost available node $\geq a_i$.
  - Can be implemented in $\mathcal{O}(n \log n)$ time with some balanced search tree, `set` is good enough.

Ringworld

- Given n intervals on a circle, choose one node inside each interval so that no node is used twice.
- First consider the problem on a line. Then a simple greedy algorithm works:
  - scan the nodes from left to right,
  - whenever we counter the right endpoint b_i of an interval $[a_i, b_i]$, choose the leftmost available node $\geq a_i$.
  - Can be implemented in $\mathcal{O}(n \log n)$ time with some balanced search tree, `set` is good enough.
- Now go back to the circle. Does the same reasoning work?

Ringworld

- If we can cut the circle in such a way that the cut doesn't intersect any interval, we can apply the above method.
- But what to do when we cannot find such a cut?

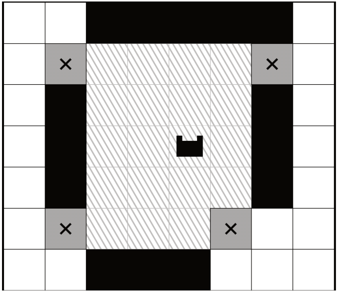
Ringworld

- If we can cut the circle in such a way that the cut doesn't intersect any interval, we can apply the above method.
- But what to do when we cannot find such a cut?
- For each interval $[a_i, b_i]$ create one or two intervals on $[0, 1, \ldots, 2m - 1]$:
  - if $a_i \leq b_i$ create $[a_i, b_i]$ and $[m + a_i, m + b_i]$,
  - otherwise create $[a_i, m + b_i]$.

Ringworld

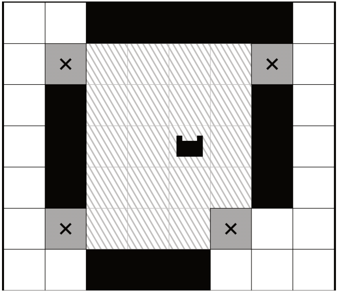
- If we can cut the circle in such a way that the cut doesn't intersect any interval, we can apply the above method.
- But what to do when we cannot find such a cut?
- For each interval $[a_i, b_i]$ create one or two intervals on $[0, 1, \ldots, 2m - 1]$:
    - if $a_i \leq b_i$ create $[a_i, b_i]$ and $[m + a_i, m + b_i]$,
    - otherwise create $[a_i, m + b_i]$.
- Now you can prove that if $n \leq m$, the original circle problem has a solution iff the new line problem has a solution.
- Proof formulating the question as a matching in a bipartite graph, and applying the Hall's theorem.

The King of the North



- Classic flow problem with vertex capacities where castle $\widehat{=}$ source and the (unshown) border $\widehat{=}$ sink

The King of the North



- Classic flow problem with vertex capacities where castle $\widehat{=}$ source and the (unshown) border $\widehat{=}$ sink
- Reduce to flow problem by using vertex duplication (in/out vertex) for the arc capacities
- Perform your standard max-flow algorithm to calculate the minimum cut

Ticket Draw

- For $M = m_1 \ldots m_n$, $Z = z_1 \ldots z_n$, and r, compute $S(n)$ – the number of strings $a_1, \ldots, a_n$ over $\{0, 1, \ldots, 9\}$ of length n which
  (1) represent integers smaller or equal to $M - 1$ and
  (2) do not r-match Z, i.e. such that
  $z_i \ldots z_{i+r-1} \neq a_i \ldots a_{i+r-1}$ for all i.
- The number of tickets is $M - S(n)$.

Ticket Draw

- For $M = m_1 \ldots m_n$, $Z = z_1 \ldots z_n$, and r, compute $S(n)$ – the number of strings $a_1, \ldots, a_n$ over $\{0, 1, \ldots, 9\}$ of length n which
  (1) represent integers smaller or equal to $M - 1$ and
  (2) do not r-match Z, i.e. such that
  $z_i \ldots z_{i+r-1} \neq a_i \ldots a_{i+r-1}$ for all i.
- The number of tickets is $M - S(n)$.
- First an easier task: drop the constraint (1) and compute $F(n)$ – the number of strings of length n which do not r-match Z.
- Use DP and the recurrence relation:

$$F(n) = \sum_{i=1}^{r} 9 \cdot F(n-i)$$

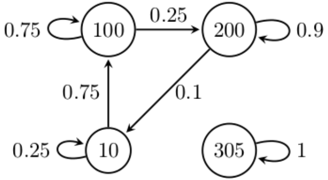
Ticket Draw

- Using $F(1), F(2), \ldots F(n)$ compute $S(n)$ via DP.
- Start with $S(1), \ldots, S(r)$ and compute $S(k)$ for $k = r + 1, \ldots, n$.
  - If $z_k > m_k$ then $S(k) = m_k \cdot F(k-1) + S(k-1)$.
  - If $z_k < m_k$ then $S(k) = (m_k - 1) \cdot F(k-1) + S(k-1) + \sum_{j=2}^{r} 9 \cdot F(k-j)$.
  - What if $z_k = m_k$?

Ticket Draw

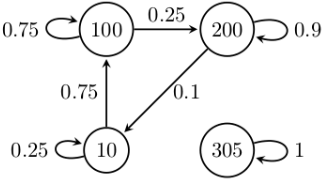
- Using $F(1), F(2), \ldots F(n)$ compute $S(n)$ via DP.
- Start with $S(1), \ldots, S(r)$ and compute $S(k)$ for $k = r+1, \ldots, n$.
  - If $z_k > m_k$ then $S(k) = m_k \cdot F(k-1) + S(k-1)$.
  - If $z_k < m_k$ then $S(k) = (m_k - 1) \cdot F(k-1) + S(k-1) + \sum_{j=2}^{r} 9 \cdot F(k-j)$.
  - What if $z_k = m_k$?
  - Idea: initialize the value $S(k)$ with $m_k \cdot F(k-1)$ and continue with comparing next digits.
- Running time: $O(r \cdot \log M)$.

Timing



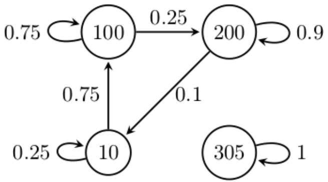
- Each node's output is ≥ 0 and sums up to 1

Timing



- Each node's output is ≥ 0 and sums up to 1
- Evolution can be modelled as a Markov chain

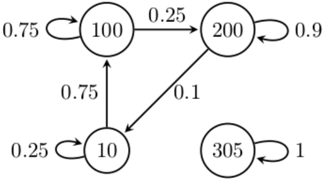
Timing



$$\begin{pmatrix} 0.75 & 0.00 & 0.75 & 0.00 \\ 0.25 & 0.90 & 0.00 & 0.00 \\ 0.00 & 0.10 & 0.25 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}^t \cdot \begin{pmatrix} 100 \\ 200 \\ 10 \\ 305 \end{pmatrix}$$

- Each node's output is ≥ 0 and sums up to 1
- Evolution can be modelled as a Markov chain
- $\Rightarrow$ compute $A^t \cdot \vec{u}$
  ($A \widehat{=}$ matrix of links, $t \widehat{=}$ time until attack, $\vec{u} \widehat{=}$ strengths)

Timing



$$\begin{pmatrix} 0.75 & 0.00 & 0.75 & 0.00 \\ 0.25 & 0.90 & 0.00 & 0.00 \\ 0.00 & 0.10 & 0.25 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}^t \cdot \begin{pmatrix} 100 \\ 200 \\ 10 \\ 305 \end{pmatrix}$$

- Each node's output is ≥ 0 and sums up to 1
- Evolution can be modelled as a Markov chain
- $\Rightarrow$ compute $A^t \cdot \vec{u}$
  ($A \mathrel{\widehat{=}}$ matrix of links, $t \mathrel{\widehat{=}}$ time until attack, $\vec{u} \mathrel{\widehat{=}}$ strengths)
- $\mathcal{O}(t \cdot N^3)$ is too slow $\Rightarrow$ use fast exponentiation
- Compute weak point by looking at the direct neighbourhood in $\mathcal{O}(N^2)$

Triangles

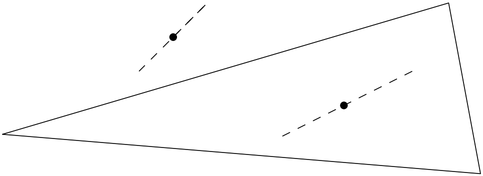
- Given two triangles *a* and *b* in 3D, are they tangled?
- The problem statement excludes all degenerate cases.
- 3D is difficult, so maybe try to reduce the problem to 2D?

Triangles

- Given two triangles a and b in 3D, are they tangled?
- The problem statement excludes all degenerate cases.
- 3D is difficult, so maybe try to reduce the problem to 2D?
- Take a and consider the (unique) plane P containing it.

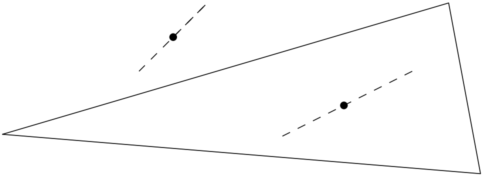
Triangles

- Draw both a and the intersection of b with P.



Triangles

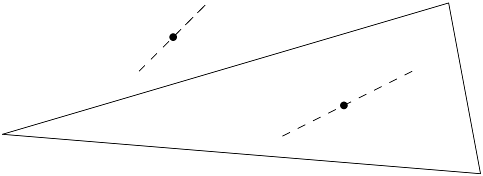
- Draw both a and the intersection of b with P.



- The triangles are tangled iff the the intersection of b with P contains a point inside and a point outside of a (on P).

Triangles

- Draw both a and the intersection of b with P.



- The triangles are tangled iff the the intersection of b with P contains a point inside and a point outside of a (on P).
- Many ways to compute the intersection, the simplest is probably solving a system of linear equations.