German Collegiate Programming Contest

GCPC Jury

gcpc@nwerc.eu

2. Juli 2011

jury sample solutions

Problem	**min. LOC**	**max. LOC**
Faculty Dividing Powers	31	54
Genetic Fraud	32	48
Indiana Jones and the lost Soccer Cup	50	60
Magic Star	49	74
Magical Crafting	46	93
My brother's diary	31	47
Security Zone	67	199
Sightseeing	34	113
Suiting Weavers	88	179
Time to live	32	65
$\sum$	460	932

Faculty Dividing Powers

- Given n and k, find $\max\left\{ i \mid k^i \text{ divides } n! \right\}$

Faculty Dividing Powers

- Given n and k, find $\max\left\{ i \mid k^i \text{ divides } n! \right\}$
- Consider prime factorizations of $n!$ and k:

$$n! = p_1^{n_1} \cdot p_2^{n_2} \cdot p_3^{n_3} \cdot \ldots$$
$$k = p_1^{k_1} \cdot p_2^{k_2} \cdot p_3^{k_3} \cdot \ldots$$

Faculty Dividing Powers

- Given n and k, find $\max\left\{ i \mid k^i \text{ divides } n! \right\}$
- Consider prime factorizations of $n!$ and k:

$$n! = p_1^{n_1} \cdot p_2^{n_2} \cdot p_3^{n_3} \cdot \ldots$$
$$k = p_1^{k_1} \cdot p_2^{k_2} \cdot p_3^{k_3} \cdot \ldots$$

- Then $k^i = p_1^{ik_1} \cdot p_2^{ik_2} \cdot p_3^{ik_3} \cdot \ldots$ and we require $ik_1 \leq n_1, ik_2 \leq n_2, ik_3 \leq n_3, \ldots$

Faculty Dividing Powers

- Given n and k, find $\max\left\{i \mid k^i \text{ divides } n!\right\}$
- Consider prime factorizations of $n!$ and k:

$$n! = p_1^{n_1} \cdot p_2^{n_2} \cdot p_3^{n_3} \cdot \ldots$$
$$k = p_1^{k_1} \cdot p_2^{k_2} \cdot p_3^{k_3} \cdot \ldots$$

- Then $k^i = p_1^{ik_1} \cdot p_2^{ik_2} \cdot p_3^{ik_3} \cdot \ldots$ and we require $ik_1 \leq n_1, ik_2 \leq n_2, ik_3 \leq n_3, \ldots$
- So solution is $\min_{i \geq 1, k_i > 0} \left\{\lfloor n_i / k_i \rfloor\right\}$

Faculty Dividing Powers

- Find prime factorization of k in time $O(\sqrt{k})$

Faculty Dividing Powers

- Find prime factorization of k in time $O(\sqrt{k})$
- For each prime factor p of k, find out how often p divides $n!$ by the following formula:

$$\left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \ldots$$

Faculty Dividing Powers

- Find prime factorization of k in time $O(\sqrt{k})$
- For each prime factor p of k, find out how often p divides $n!$ by the following formula:

$$\left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \ldots$$

- Number of prime factors is $O(\log k)$, evaluating the formula takes time $O(\log n)$ for each one

Faculty Dividing Powers

- Find prime factorization of k in time $O(\sqrt{k})$
- For each prime factor p of k, find out how often p divides $n!$ by the following formula:

$$\left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \ldots$$

- Number of prime factors is $O(\log k)$, evaluating the formula takes time $O(\log n)$ for each one
- So whole runtime is $O(\sqrt{k} + \log k \cdot \log n)$
- Mistakes:
  - Scanner.nextInt()
  - while(ktmp¡=n) vs. while(ktmp¡=n/primes[m].first)

Faculty Dividing Powers

- Example: $n = 15$, $k = 12$
- $k = 12 = 2^2 3^1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2^1		x		x		x		x		x		x		x	
2^2				x				x				x			
2^3								x							
3^1			x			x			x			x			x
3^2									x						

- $n!$ contains $2^{11}3^6$
- $i = min(\frac{11}{2}, \frac{6}{1}) = 5$

Genetic Fraud

- Gegeben sind zwei Strings aus Kleinbuchstaben
- Die Frage ist ob eine Subsequenz der Länge L/2 oder r existiert, bei der leichte Fehler (+/- 1) erlaubt sind

Genetic Fraud

- Gegeben sind zwei Strings aus Kleinbuchstaben
- Die Frage ist ob eine Subsequenz der Länge L/2 oder r existiert, bei der leichte Fehler (+/- 1) erlaubt sind
- Zur Lösung reicht das Standardverfahren für LCS (dynamische Programmierung) ebenfalls aus
- Problematisch konnte hauptsächlich Division durch zwei werden
- durch fehlerhaften Testinput: Greedy wurde auch akzeptiert

Indiana Jones

- Standard topological sort

Indiana Jones

- Standard topological sort
- Remove nodes with indegree 0 one by one, until there's none left (including out-going edges)

Indiana Jones

- Standard topological sort
- Remove nodes with indegree 0 one by one, until there's none left (including out-going edges)
- If not all nodes have been removed, output "`recheck hints`"

Indiana Jones

- Standard topological sort
- Remove nodes with indegree 0 one by one, until there's none left (including out-going edges)
- If not all nodes have been removed, output "`recheck hints`"
- If all nodes have been removed, but at one point there was more than one node with indegree 0, output "`missing hints`"

Indiana Jones

- Standard topological sort
- Remove nodes with indegree 0 one by one, until there's none left (including out-going edges)
- If not all nodes have been removed, output "`recheck hints`"
- If all nodes have been removed, but at one point there was more than one node with indegree 0, output "`missing hints`"
- Else output nodes in the order they were removed

Indiana Jones

- Standard topological sort
- Remove nodes with indegree 0 one by one, until there's none left (including out-going edges)
- If not all nodes have been removed, output "`recheck hints`"
- If all nodes have been removed, but at one point there was more than one node with indegree 0, output "`missing hints`"
- Else output nodes in the order they were removed
- common mistake: stopped immediately once more than one node with indegree 0 was available, instead of checking whether all nodes will be removed

Magic Star

- Simple backtracking runs in time.
- There are fewer than 12^8 solutions (when you place three values in a row, the fourth is determined).
- The real number of possible solutions is considerably smaller.

Magic Star

- Simple backtracking runs in time.
- There are fewer than 12^8 solutions (when you place three values in a row, the fourth is determined).
- The real number of possible solutions is considerably smaller.
- Construct all possible solutions and select the lexicographically smallest one.

Magic Star

- Simple backtracking runs in time.
- There are fewer than 12^8 solutions (when you place three values in a row, the fourth is determined).
- The real number of possible solutions is considerably smaller.
- Construct all possible solutions and select the lexicographically smallest one.
- Better: construct the solutions in lexicographical order, then you can stop as soon as the first solution is found.

Magical Crafting

- Gegeben ist ein Set von Rezepten und ein Leuchteffekt
- Herauszufinden ist, ob ein Leuchteffekt aus den Rezepten erstellt werden kann . . .
- . . . und wenn ja, mit welchen Kosten

Magical Crafting

- Gegeben ist ein Set von Rezepten und ein Leuchteffekt
- Herauszufinden ist, ob ein Leuchteffekt aus den Rezepten erstellt werden kann . . .
- . . . und wenn ja, mit welchen Kosten
- Die Rezepte sind Regeln einer Grammatik in (beinahe) Chomsky Normalform
- Der CYK Algorithmus berechnet die Möglichkeit
- Erweitert um den Kostenfaktor liefert er auch die minimale Anzahl Diamanten

My brother's diary

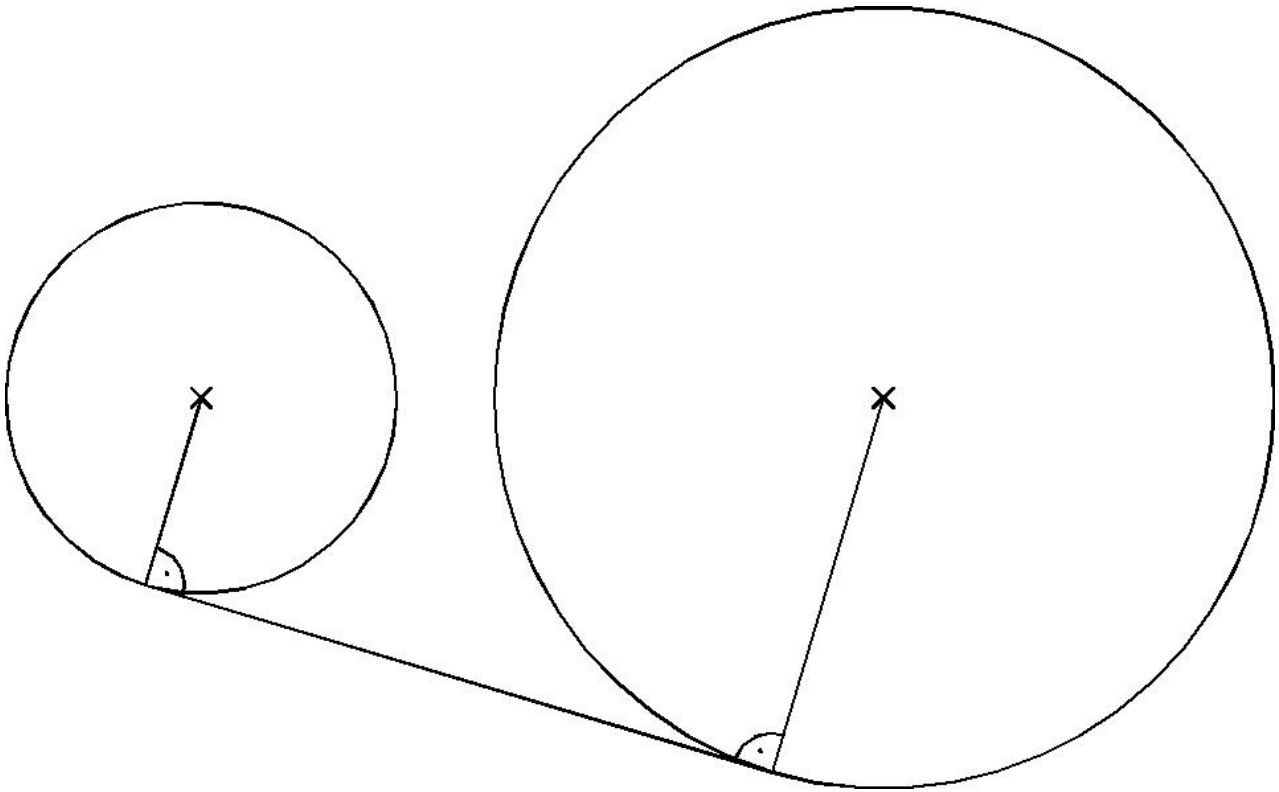
- No-Brainer
- count letter frequencies
- is the most frequent letter unique?
- shift distance d is calculated by $(26 + (maxChar - \text{'E'}))\%26$

My brother's diary

- No-Brainer
- count letter frequencies
- is the most frequent letter unique?
- shift distance d is calculated by $(26 + (maxChar - 'E'))\%26$
- calculated correct distance d, but not in interval $0 \leq d \leq 25$

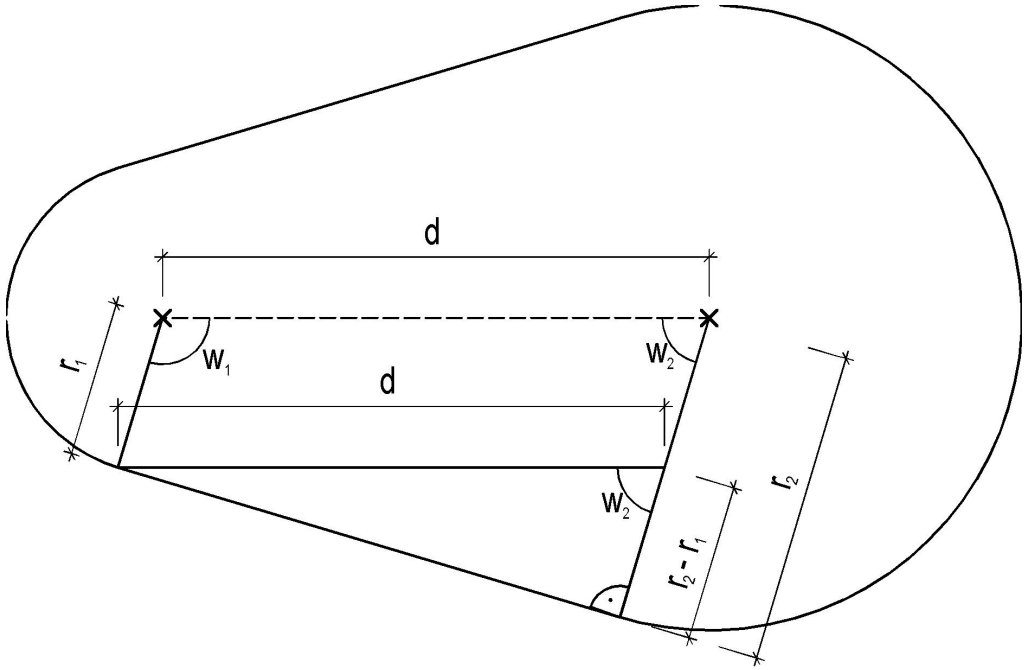
Security Zone

- reduction to two circles:



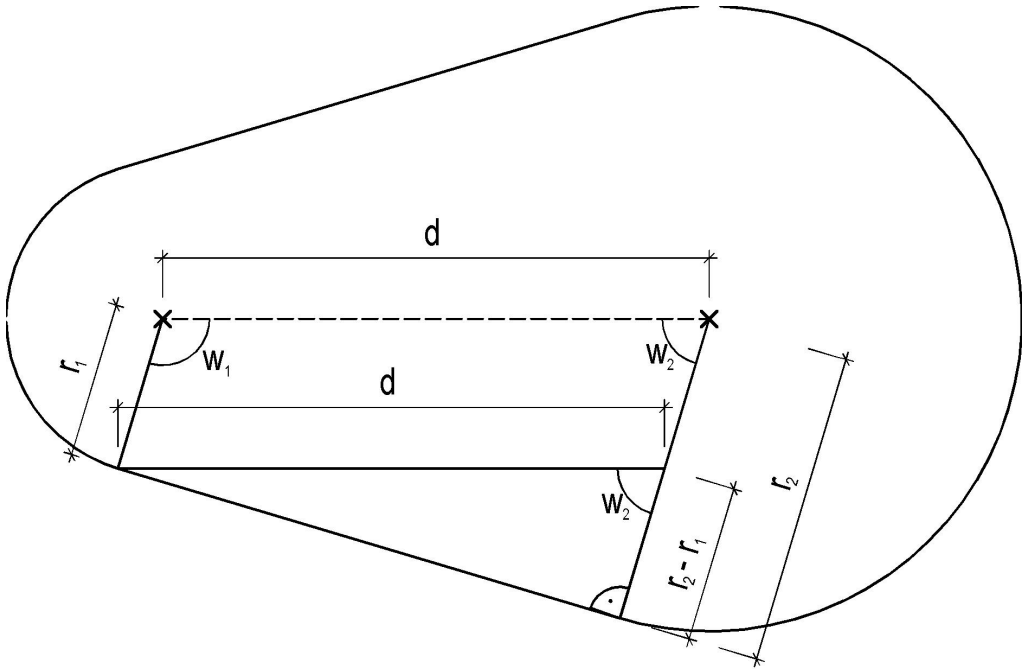
Security Zone

- reduction to two circles:



Security Zone

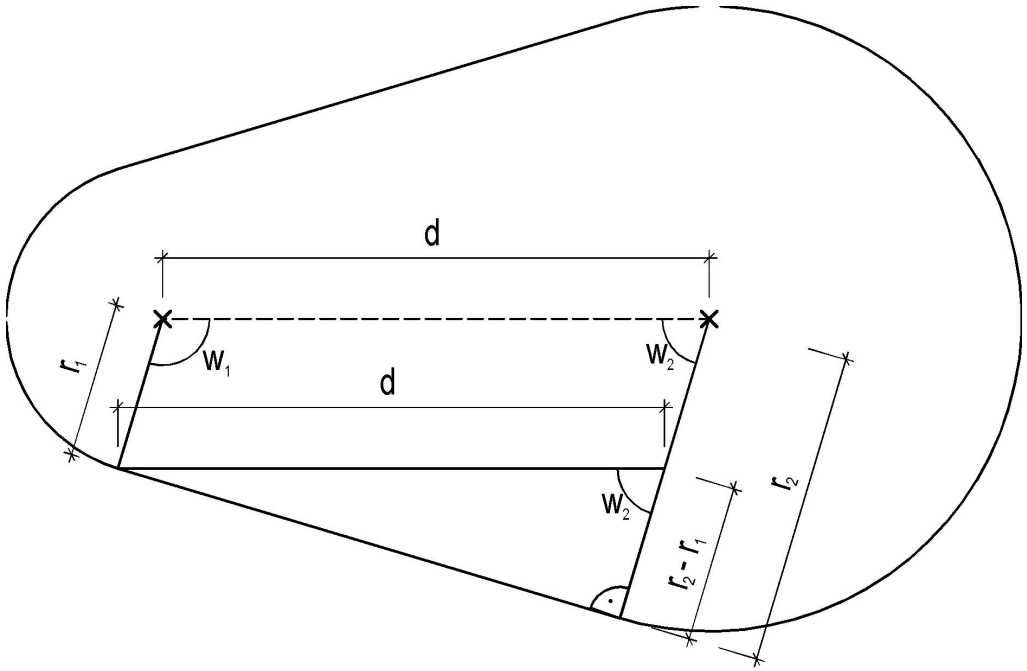
- reduction to two circles:



- $w_2 = \arccos(\frac{r_2 - r_1}{d})$
- $w_1 = \pi - \arccos(\frac{r_2 - r_1}{d})$

Security Zone

- reduction to two circles:



- $w_2 = \arccos(\frac{r_2 - r_1}{d})$
- $w_1 = \pi - \arccos(\frac{r_2 - r_1}{d}) = \arccos(\frac{r_1 - r_2}{d})$

Security Zone

- algorithm for arbitrary number of circles:
  (compute convex hull by wrapping around circles)

Security Zone

- algorithm for arbitrary number of circles:
  (compute convex hull by wrapping around circles)
- search start point (circle + angle)

Security Zone

- algorithm for arbitrary number of circles:
  (compute convex hull by wrapping around circles)
- search start point (circle + angle)
- while not finished:
  - iterate over other every circle and search for next angle / transition to other circle

Security Zone

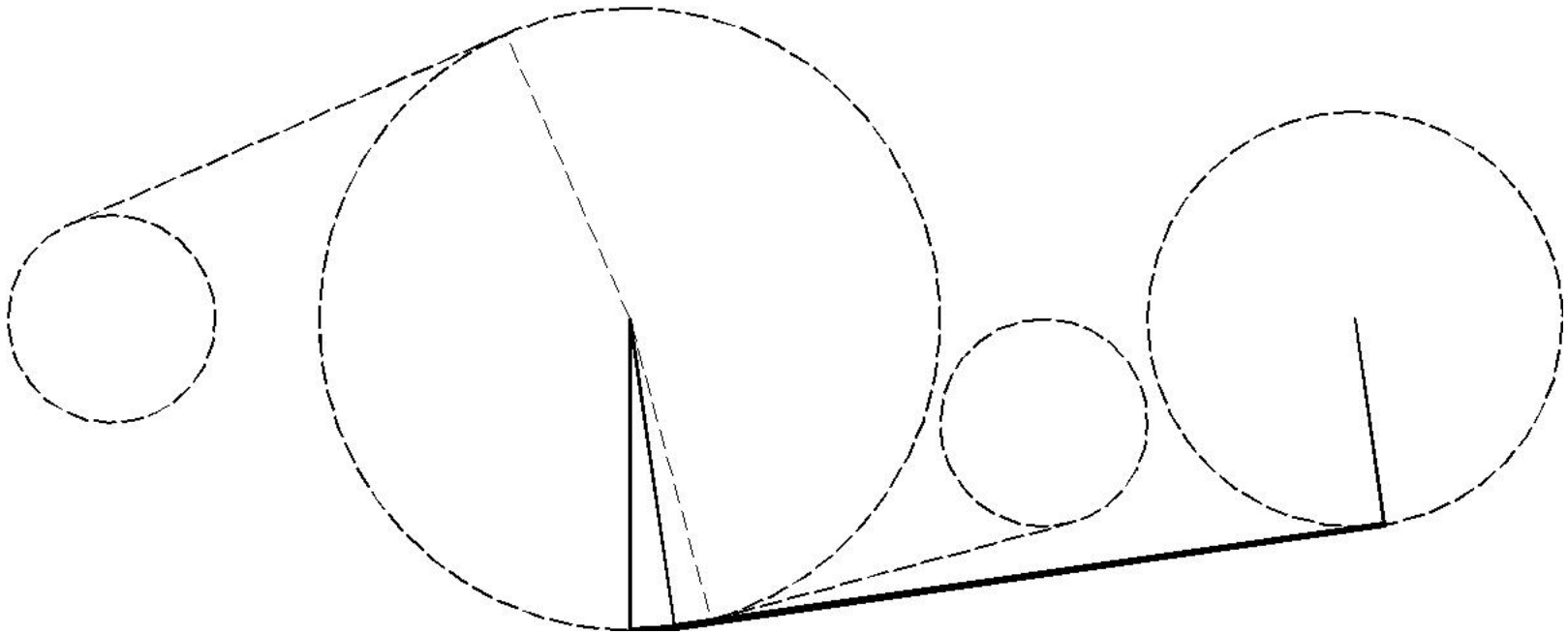
- algorithm for arbitrary number of circles:
  (compute convex hull by wrapping around circles)
- search start point (circle + angle)
- while not finished:
  - iterate over other every circle and search for next angle / transition to other circle
  - add radial part between last angle and next angle

Security Zone

- algorithm for arbitrary number of circles:
  (compute convex hull by wrapping around circles)
- search start point (circle + angle)
- while not finished:
  - iterate over other every circle and search for next angle / transition to other circle
  - add radial part between last angle and next angle
  - add distance between the two points on the corresponding two circles

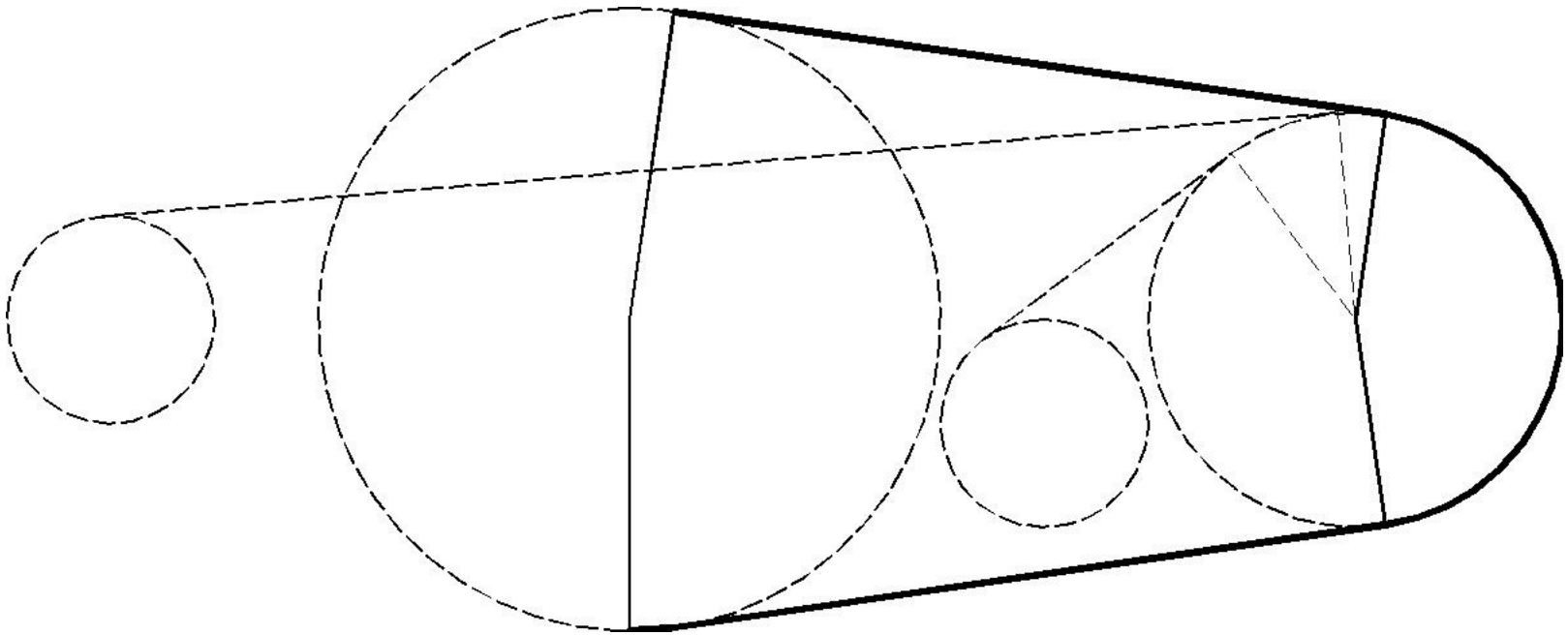
Security Zone

- example:



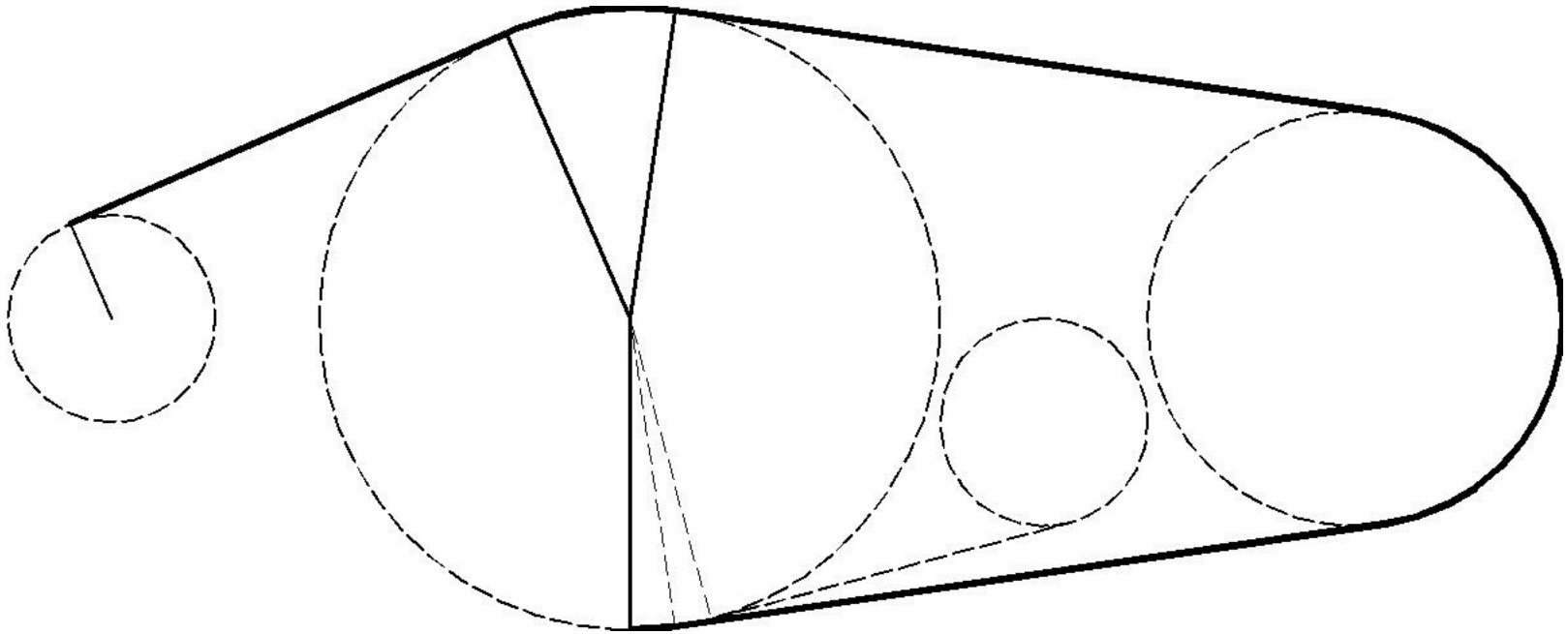
Security Zone

- example:



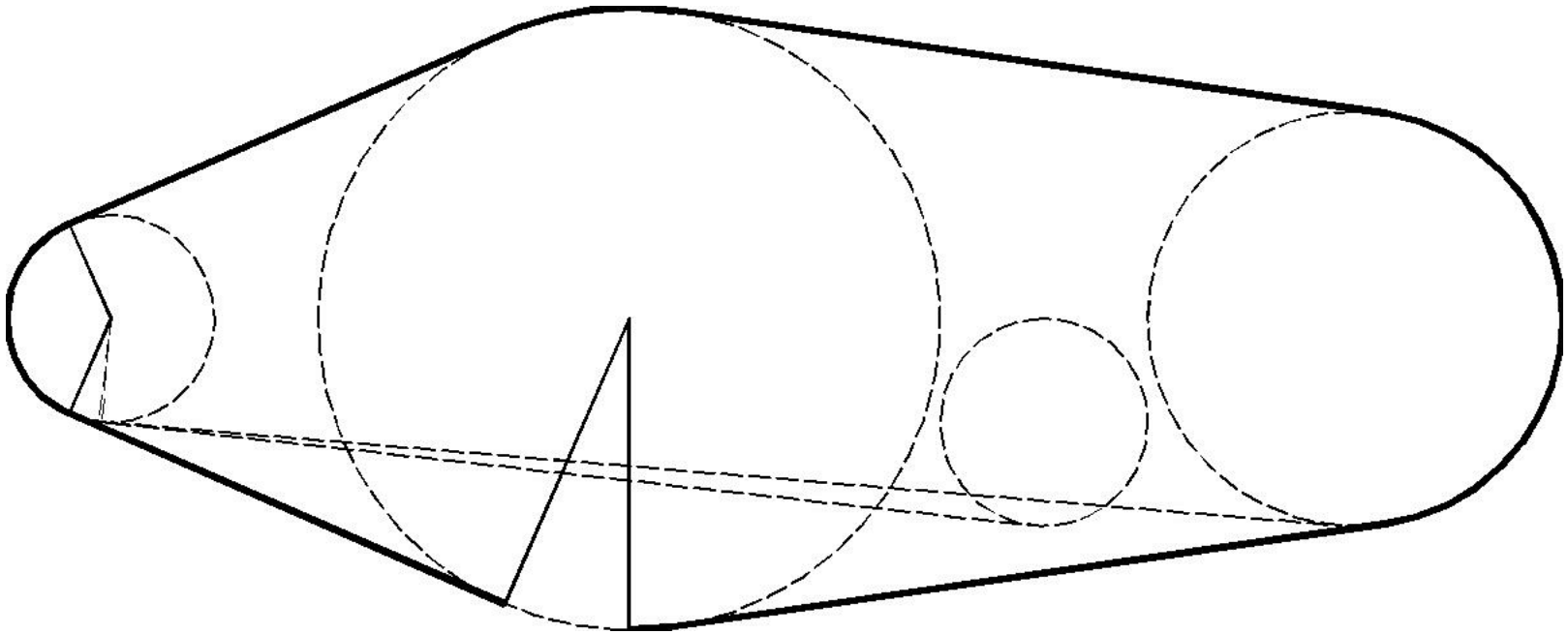
Security Zone

- example:



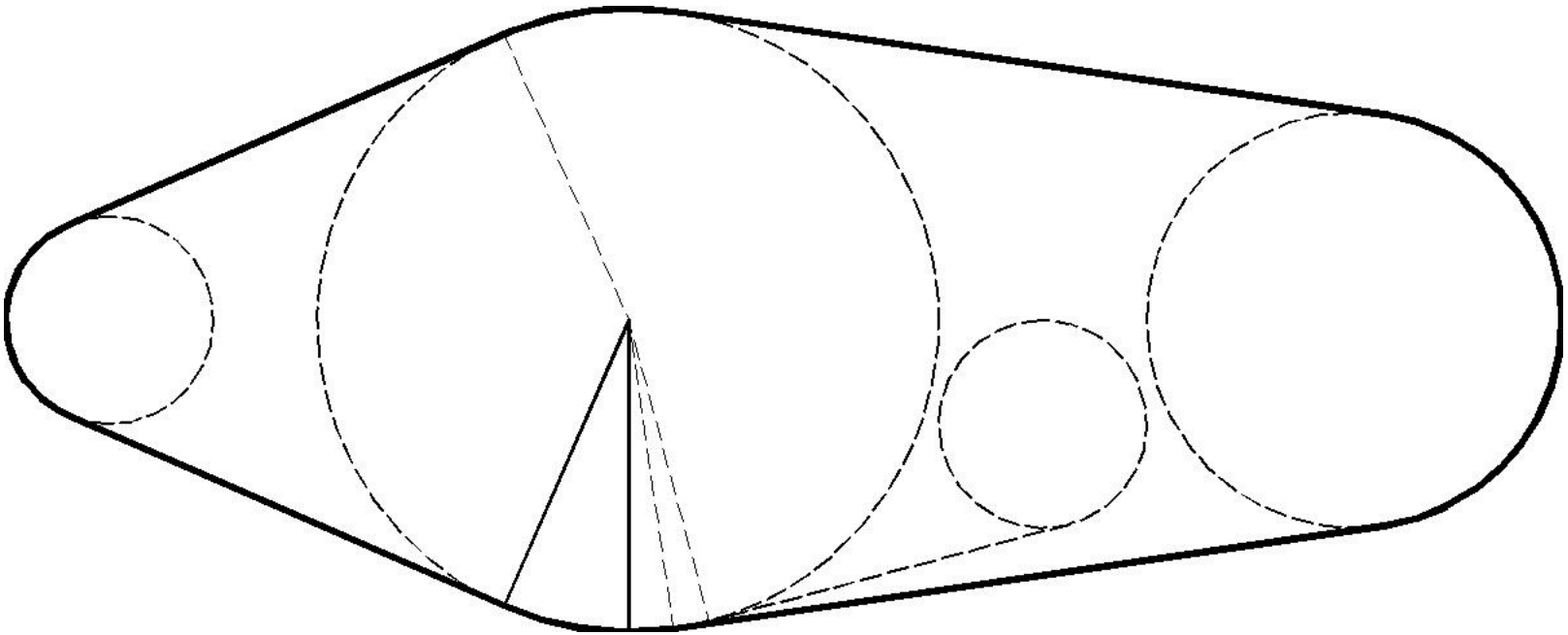
Security Zone

- example:



Security Zone

- example:



Security Zone

- common mistake: you did not submit

Sightseeing

- Gesucht ist eine kürzeste Route über ein zirkuläres Set von Strecken
- Keine der Strecken darf dabei mehrfach durchlaufen werden

Sightseeing

- Gesucht ist eine kürzeste Route über ein zirkuläres Set von Strecken
- Keine der Strecken darf dabei mehrfach durchlaufen werden
- Bei bekanntem Startpunkt lässt sich der kürzeste Weg in Linearzeit berechnen (Simple Graphstruktur)
- $\Rightarrow$ Simulation durch Dynamische Programmierung
- Die Strecke musste vom Ziel her rekonstruiert werden
- Auch ausreichend: Dijkstra Algorithmus
- Integer Überläufe konnten richtige Zeit voräuschen

Suiting Weavers

Problem

Possible to assign fibers to weavers, so that no weaver has more fibers than Willy?

General Approach

- Willy picks up all fibers he can reach
- Calculate maximum fibers that each weaver may pick up not surpassing Willy
- Use maximum flow algorithm to decide

Suiting Weavers (2)

Assume Willy collects all fibers in reach

- Find these places
- Assign fibers to Willy
- This gives Willy's optimum number of fibers W

Shortcut: If there exists a weaver with more than W fibers $\Rightarrow$ `Lonesome Willy`

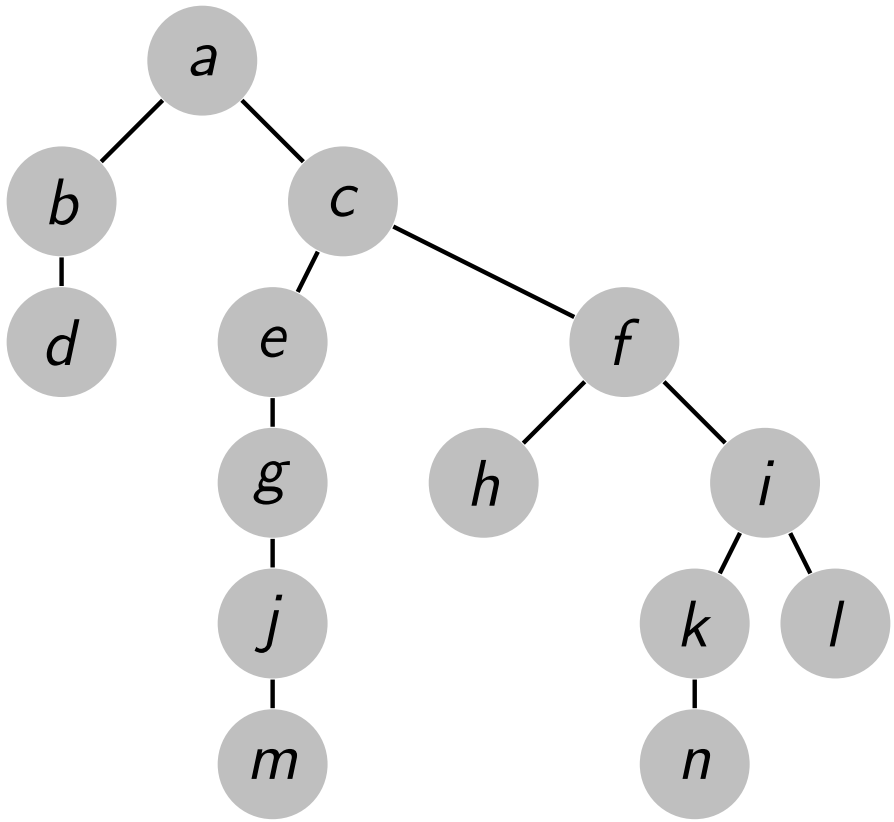
Suiting Weavers (3)

Maximum flow network

- One node for each remaining place and weaver, an additional source and a sink
- Edges
  - From each place to weavers that can reach it
    Capacity: number of fibers (or ∞)
  - From source to each place
    Capacity: number of fibers of the place
  - From all weavers to sink
    Capacity: difference of W and the initial number of fibers collected by the corresponding weaver
- max. flow $= \sum f_i \quad \Leftrightarrow \quad$ `Suiting Success`
  i.e., all remaining fibers were assigned

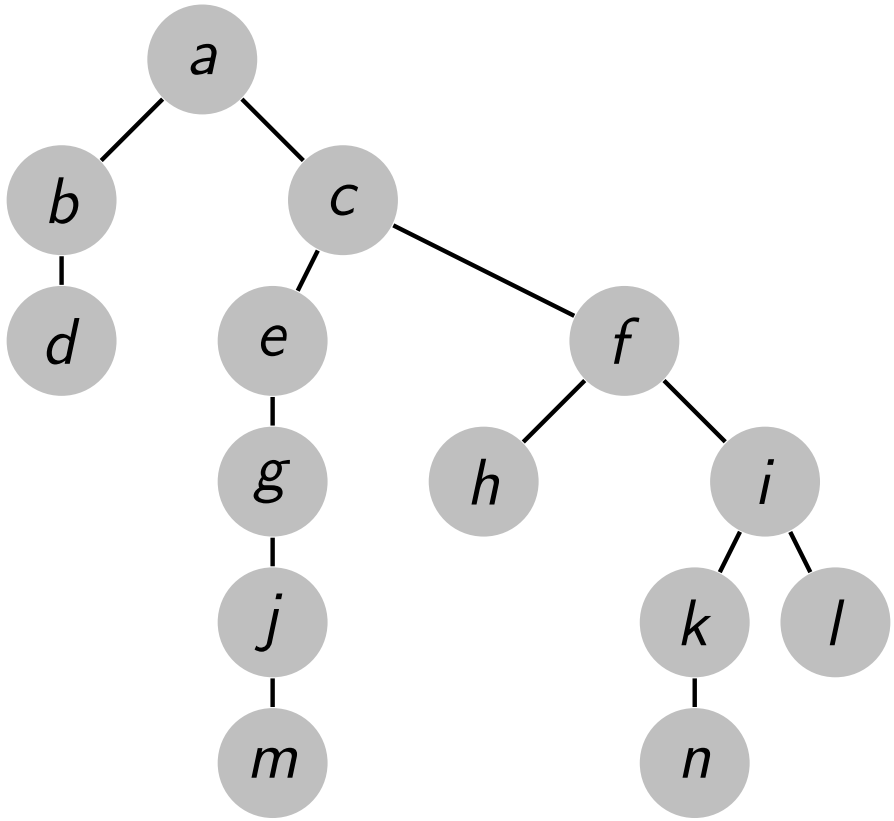
Time To Live

- given: tree
- find: node X (resp. path length) – the path length from any node in the tree to X should be minimized



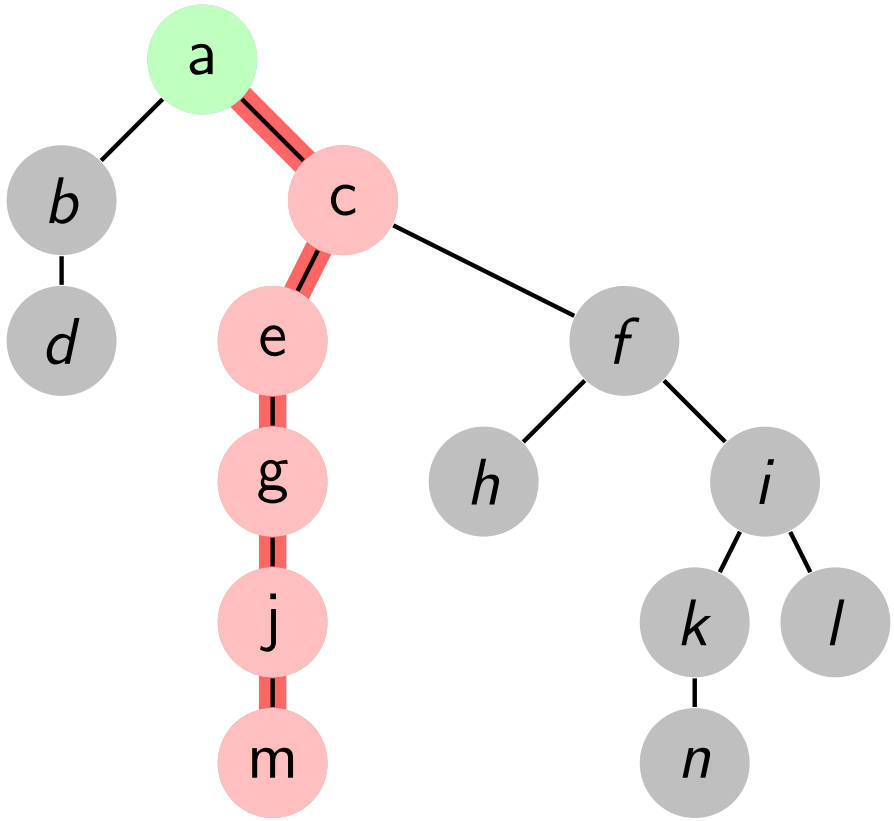
Time To Live

- $\Rightarrow$ compute length l of longest path in tree, answer: $(l+1)/2$
- do BFS from arbitrary root first, then do BFS from last node in previous BFS



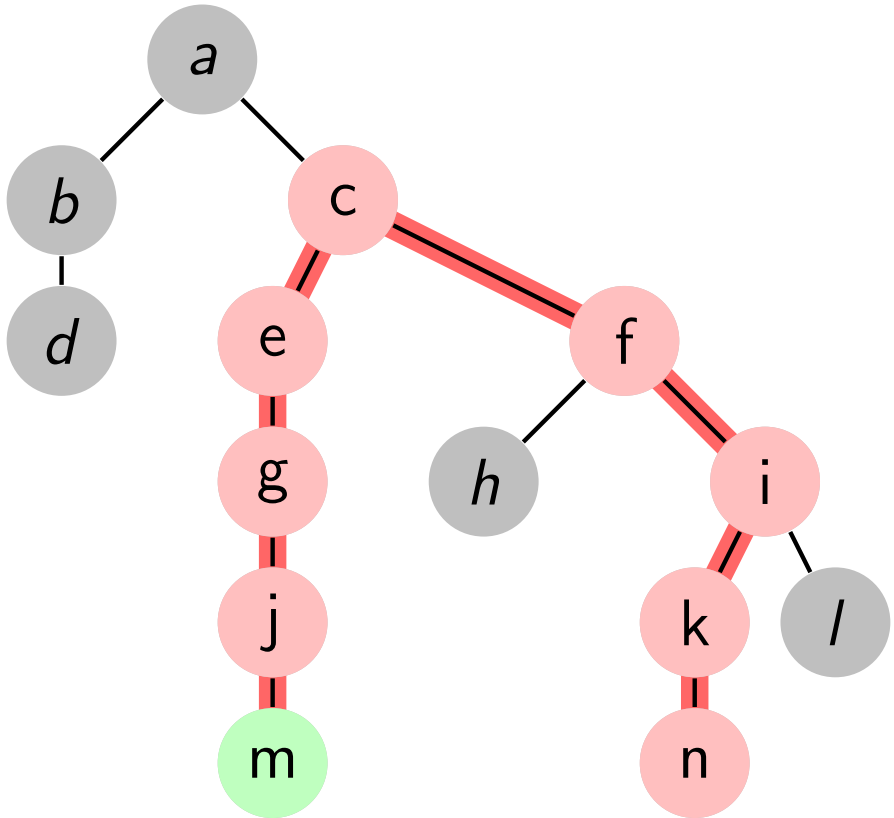
Time To Live

- $\Rightarrow$ compute length l of longest path in tree, answer: $(l+1)/2$
- do BFS from arbitrary root first, then do BFS from last node in previous BFS



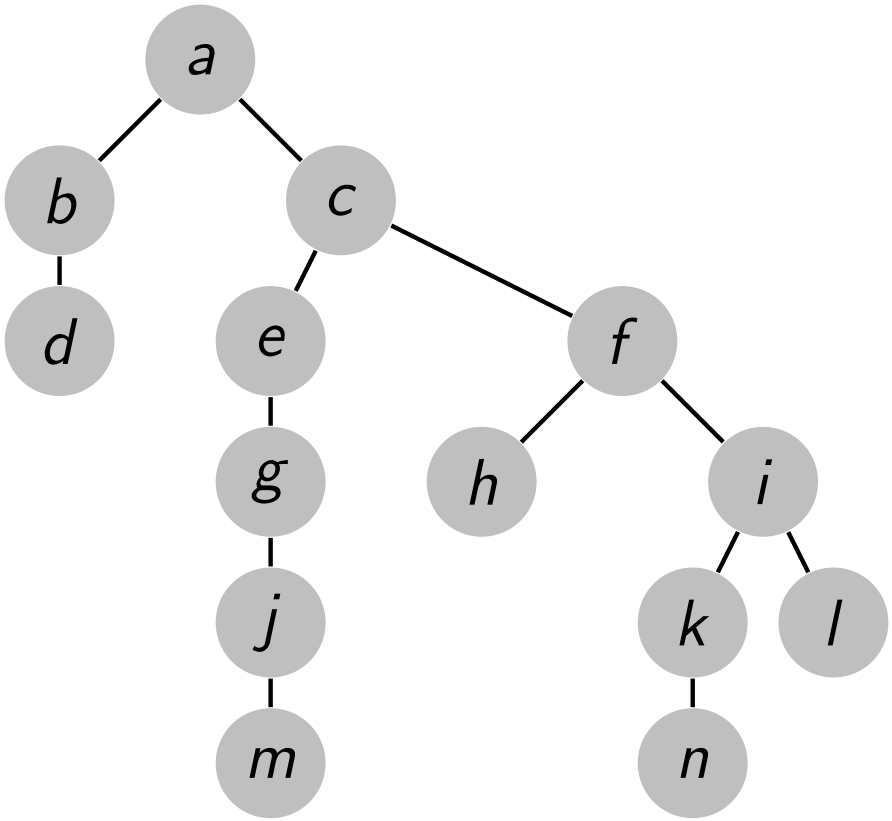
Time To Live

- $\Rightarrow$ compute length l of longest path in tree, answer: $(l+1)/2$
- do BFS from arbitrary root first, then do BFS from last node in previous BFS



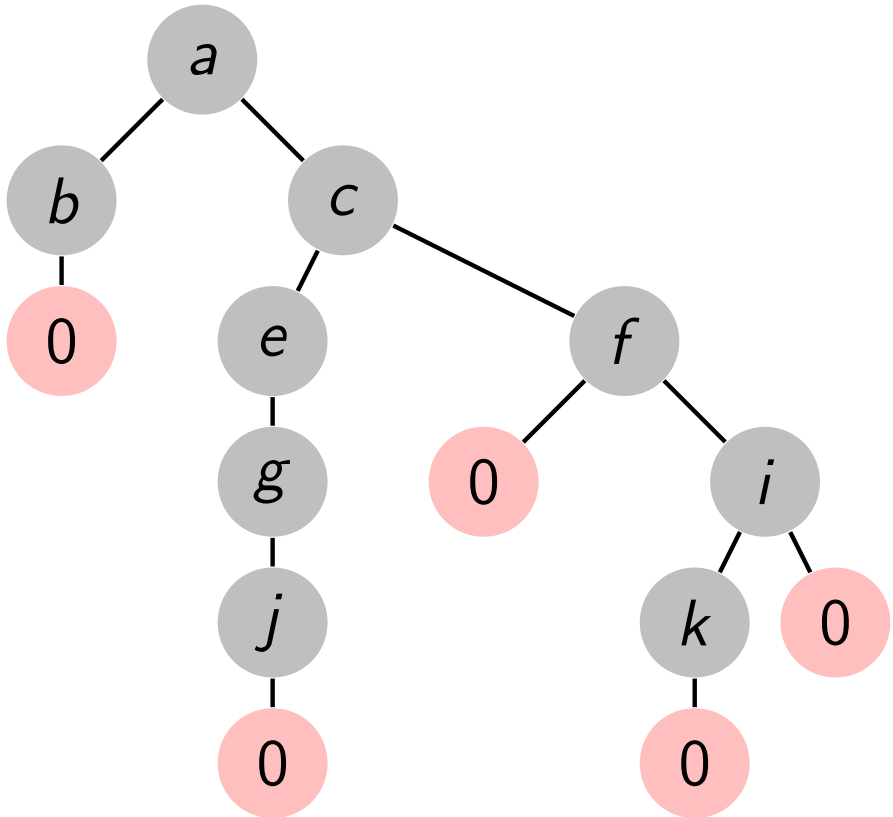
Time To Live

- alternative: number „layers“ in tree
- leafs on layer 0, other nodes on layer $1 + max(successors)$



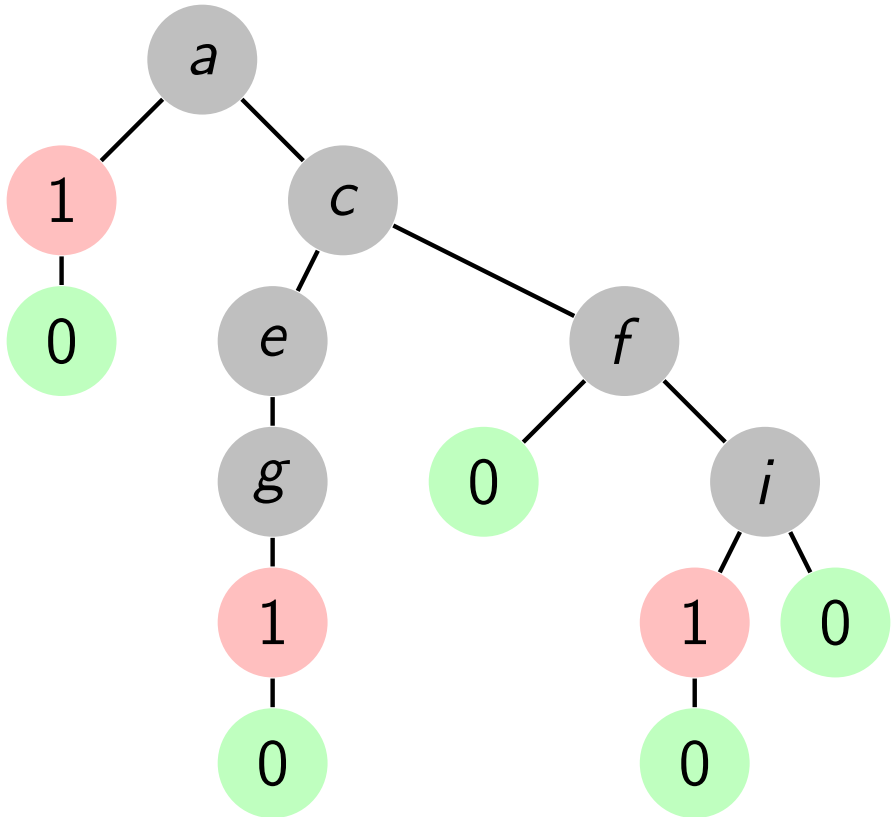
Time To Live

- alternative: number „layers“ in tree
- leafs on layer 0, other nodes on layer $1 + max(successors)$



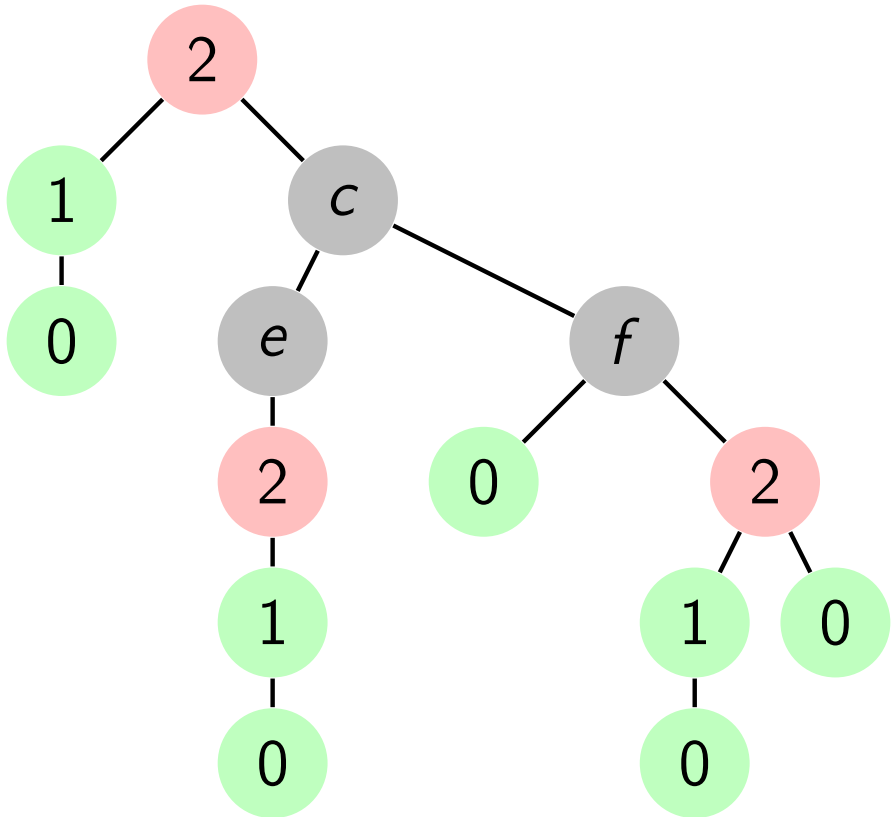
Time To Live

- alternative: number „layers“ in tree
- leafs on layer 0, other nodes on layer $1 + max(successors)$



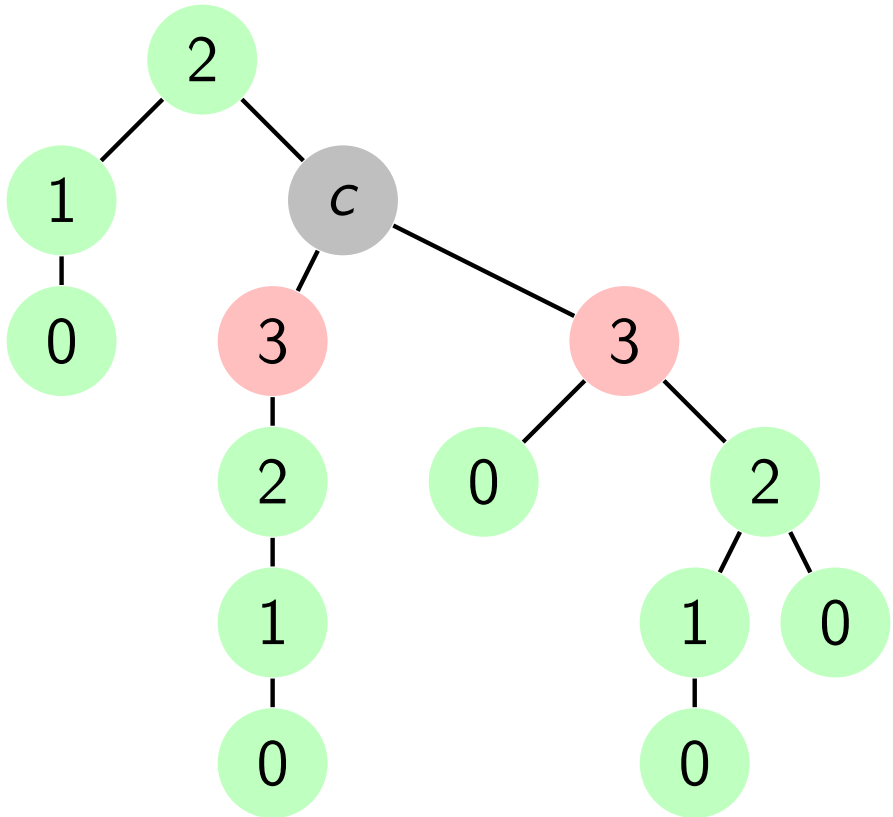
Time To Live

- alternative: number „layers“ in tree
- leafs on layer 0, other nodes on layer $1 + max(successors)$



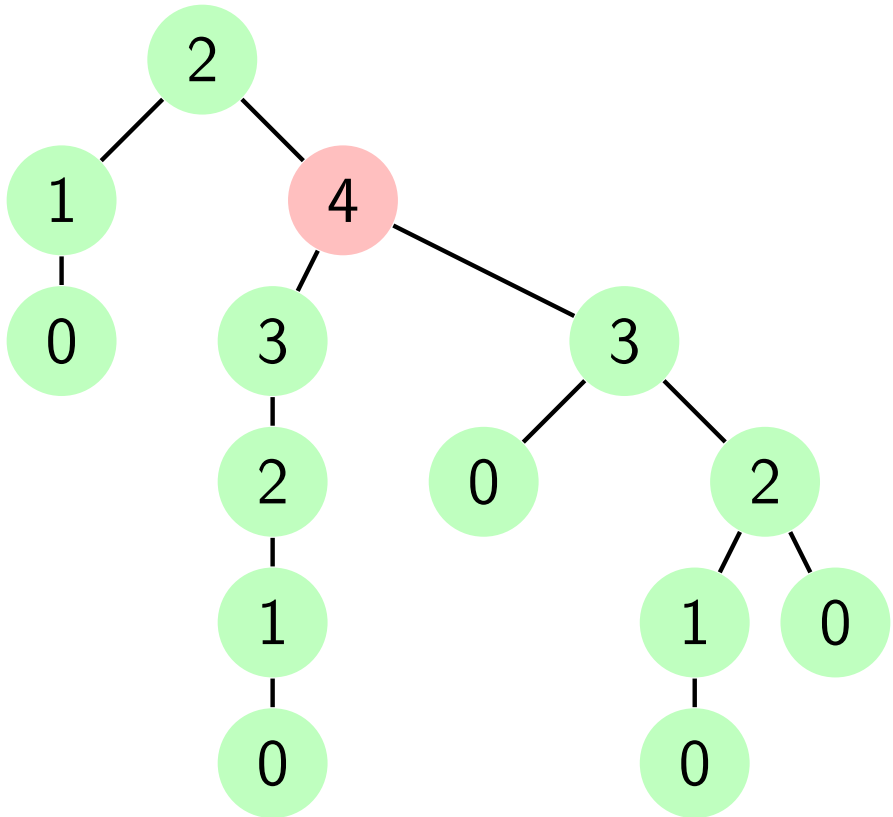
Time To Live

- alternative: number „layers“ in tree
- leafs on layer 0, other nodes on layer $1 + max(successors)$



Time To Live

- alternative: number „layers“ in tree
- leafs on layer 0, other nodes on layer $1 + max(successors)$



Award Ceremony