2017 ACM-ICPC North America Qualification Contest Solution Outlines

The Judges

October 7, 2017

North America Qualifier 2017
acm International Collegiate Programming Contest

I – Odd Gnome – First solved at 5 minutes

Problem

Find an integer that is out of sequence.
Example: 4 5 6 10 7 8

I – Odd Gnome – First solved at 5 minutes

Problem

Find an integer that is out of sequence.

Example: 4 5 6 10 7 8

I – Odd Gnome – First solved at 5 minutes

Problem

Find an integer that is out of sequence.
Example: 4 5 6 10 7 8

Solution

- Read in each sequence (one sequence per line), and identify the first number that is not exactly one greater than previous (the King).
- Can be done line-by-line or token-by-token.
- Must continue reading all input on a line appearing after the King.

F – GlitchBot – First solved at 8 minutes

Problem

Given a sequence of instructions (Left, Right, Forward) and a target (x, y). Identify and fix the one wrong instruction so that the sequence goes to the target (starting at the origin).

F – GlitchBot – First solved at 8 minutes

Problem

Given a sequence of instructions (Left, Right, Forward) and a target (x, y). Identify and fix the one wrong instruction so that the sequence goes to the target (starting at the origin).

Solution

Try each possible (single) modification to the instruction sequence until you find one that works.

F – GlitchBot – First solved at 8 minutes

Solution Strategy – $O(n^2)$

- Given sequence $s_1, \ldots, s_n$ and target (x, y)
- For each s_i:
  - For each $j \in (\text{Left}, \text{Right}, \text{Forward})$:
    - If $j = s_i$: continue
    - Let $t \leftarrow s_i, s_i \leftarrow j$
    - Check if s leads to the target by re-running all instructions. If so, report that and quit.
    - Let $s_i \leftarrow t$

F – GlitchBot – First solved at 8 minutes

Solution Strategy – $O(n)$

- Same as the $O(n^2)$ strategy, except do not re-run the entire sequence for each potential modification.
- Precompute each position p_i after executing up to instruction s_i.
- For each potential change $s_i \to j$, use translation/rotation rules to figure out (in $O(1)$) how the end location would move based on the current position p_i and the original end position p_n.

J – Progressive Scramble – First solved at 11 minutes

Problem

Encrypt or decrypt messages based on a given algorithm.

J – Progressive Scramble – First solved at 11 minutes

Problem

Encrypt or decrypt messages based on a given algorithm.

Solution

Encryption: follow the algorithm.
Decryption: invert the algorithm.

J – Progressive Scramble – First solved at 11 minutes

Solution Strategy – Encryption

- Given message symbols $d_1, \ldots, d_n$.
- Compute values by ASCII arithmetic (e.g. $f(x) = x -$ 'a' $+ 1$ for letters) or by a lookup table.
- Letter i is encrypted as $h_i = (f(d_i) + h_{i-1}) \bmod 27$.
- Then replace e_i with its corresponding symbol. Also, $h_1 = f(d_1)$.

Solution Strategy – Decryption

- Given message symbols $e_1, \ldots, e_n$.
- Process similar to encryption, starting with the first symbol.
- Now $o_i = f(e_i) - f(e_{i-1})$ with an appropriate base case of $o_i = f(e_1)$.
- Then replace o_i with its corresponding symbol s_i.

B – Bumped! – First solved at 13 minutes

Problem

Given a 2-layered graph of cities (nodes) with roads (undirected edges with some cost) and flights (directed edges with zero cost), find the lowest cost path from departure city (source node) to destination city (target node) with the restriction that the lowest cost path include *at most one* flight (edge)!

B – Bumped! – First solved at 13 minutes

Problem

Given a 2-layered graph of cities (nodes) with roads (undirected edges with some cost) and flights (directed edges with zero cost), find the lowest cost path from departure city (source node) to destination city (target node) with the restriction that the lowest cost path include *at most one* flight (edge)!

Solution

- Variation of single-source shortest path problem.
- Can use Dijkstra's algorithm [1] either on modified graph or by slightly adapting it to express the restriction of no more than one flight.

B – Bumped! – First solved at 13 minutes

Strategy 1: Graph Modification

- Modify graph by making two clones of each city: c_i and c_i', representing "before" and "after" taking the only possible flight.
- Connect nodes in both subgraphs $\{c_i\}$ and $\{c_i'\}$ according to road network's edges.
- Add flight edges $c_i \to c_j'$ with cost 0 for all possible flights $i \to j$.
- Run (standard) Dijkstra on modified graph, starting with departure city c_s and stopping at c_t or c_t', whichever is encountered first.
- Answer is $\min(D_{min}(c_t), D_{min}(c_t'))$.
- Complexity: depends on Dijkstra's implementation, typically $O(|E| \log |E|)$ where $|E| = 2m + f$.

B – Bumped! – First solved at 13 minutes

Strategy 2: Adapt Dijkstra

- Adapt lazy version of Dijkstra's algorithm so that when maintaining the frontier (or fringe) of nodes to which paths have been discovered, record in a flag whether these paths include a taking a flight or not.
- When relaxing edges representing flights, set the flag.
- Do not relax flight edges if the flag is already true.
- Copy the flag when relaxing road edges.
- Complexity: same as strategy 1.

G – Greeting Card – First solved at 15 minutes

Problem

Given $n \leq 100\,000$ integer lattice points, identify how many pairs of them are exactly $2\,018$ units apart.

G – Greeting Card – First solved at 15 minutes

Problem

Given $n \leq 100\,000$ integer lattice points, identify how many pairs of them are exactly 2 018 units apart.

Solution

The simplest solution relies on the insight that two points on the integer lattice can only be exactly 2 018 units apart if they differ by either:

- 2 018 in x or y
- 1 118 and 1 680 in x and y
  - $2\,018 = \sqrt{1\,118^2 + 1\,680^2}$
  - no other integer pair works

G – Greeting Card – First solved at 15 minutes

Solution Strategy

Create a hash table with all input points and look for all neighbors in the table and count them.

There are only 12 such neighbors for any (x, y).

- $(x+2\,018, y), (x-2\,018, y), (x, y+2\,018), \ldots, (x+1\,118, y+1\,680), \ldots$

With a decent hash table, this is $O(n)$.

C – Canonical Coin Systems – First solved at 16 minutes

Problem

Given a set of coin denominations $c_1, \ldots, c_n$ ($n \leq 100$), where $c_i \leq 10^6$, is it *canonical*? I.e., does making change greedily always produce optimal number of coins?

C – Canonical Coin Systems – First solved at 16 minutes

Problem

Given a set of coin denominations $c_1, \ldots, c_n$ ($n \leq 100$), where $c_i \leq 10^6$, is it *canonical*? I.e., does making change greedily always produce optimal number of coins?

Solution

- Find a *counterexample* (an amount where the greedy algorithm fails).
- Implement both greedy and dynamic programming; search all possibilities.
- If a counterexample exists, it is less than the the sum of the two largest denominations.

C – Canonical Coin Systems – First solved at 16 minutes

Solution Strategy – Greedy

Let's make change for amount v.

Greedy(v):

- $n \leftarrow 0$
- while $v > 0$:
  - choose $c_i \leq v$ as the largest-denomination coin
  - $v \leftarrow v - c_i$
  - $n \leftarrow n + 1$
- return n

C – Canonical Coin Systems – First solved at 16 minutes

Solution Strategy – Dynamic Programming

Let $T(v, i)$ be the optimal number of coins to give change v using only coins $c_1, \ldots, c_i$.

$$T(v, i) = \min(T(v - c_i, i) + 1, T(v, i - 1))$$

Either use coin c_i or not; take the minimum of the two.

Initialize with $T(0, i) = 0$ for all i.
For each $i = 1, 2, \ldots, n$:

- For each $v = 1, 2, \ldots, c_{n-1} + c_n$:
  - Compute $T(v, i)$

Putting it together

Fill in the DP table $T(v, i)$.

For each $v \leq c_{n-1} + c_n$, compare Greedy(v) with $T(v, n)$.

If they differ in the number of coins, we have found a counterexample.

Can also be done via memoization (top-down dynamic programming).

Complexity: $O(mn)$, where $m = c_{n-1} + c_n$.

H – Imperfect GPS – First solved at 21 minutes

Problem

Given:

- Polyline R (a running path) with an arrival a_i time for each vertex v_i, and
- an interval of time t.

Find another polyline G (a GPS-estimated path) which intersects R every t seconds (and connect G and R at the beginning and end).

Then report the relative loss of distance that comes from approximating R with G.

H – Imperfect GPS – First solved at 21 minutes

Problem

Given:

- Polyline R (a running path) with an arrival a_i time for each vertex v_i, and
- an interval of time t.

Find another polyline G (a GPS-estimated path) which intersects R every t seconds (and connect G and R at the beginning and end).

Then report the relative loss of distance that comes from approximating R with G.

Solution

Read in R and the arrival times. Then for each time interval, find its position on R to get the next vertex for G. Compute the distances for both R and G and report the difference.

H – Imperfect GPS – First solved at 21 minutes

Solution Strategy

- Read in each vertex v_i and associated arrival time a_i for R.
- Precompute the total length d_r of R.
- Let $p \leftarrow v_1$ be the initial GPS position.
- Let $d_g \leftarrow 0$.
- For each time $g \in \{0, t, 2t, 3t, \ldots, a_n\}$
  - Find the next vertex v_i having $g < a_i$.
  - Geometry: compute p', the runner's position at time g, between v_{i-1} and v_i.
  - Let $d_g \leftarrow d_g + ||p - p'||$
  - Let $p \leftarrow p'$.
- Report $(d_r - d_g)/d_r$.

Runtime: $O(n + a_n)$.

L – Suspension Bridges – First solved at 21 minutes

Problem

Consider a catenary (a free-hanging rope or chain) attached to two points of the same vertical height separated by horizontal distance d. How long does the catenary have to be to achieve a given desired sag s at its lowest point at the midpoint between the attachment points?

L – Suspension Bridges – First solved at 21 minutes

Problem

Consider a catenary (a free-hanging rope or chain) attached to two points of the same vertical height separated by horizontal distance d. How long does the catenary have to be to achieve a given desired sag s at its lowest point at the midpoint between the attachment points?

Solution

Use bisection method (binary search) to find numeric root of transcendental function.

L – Suspension Bridges – First solved at 21 minutes

Solution Strategy

- Define $f(a)$ as

$$f(a) = a + s - a\cosh\frac{d}{2a}$$

  $f(a)$ is strictly monotonic for $a > 0$.
  Find $a' > 0$ such that $f(a') = 0$ and output $L(a') = 2a'\sinh\frac{d}{2a'}$.
- Extreme case is
  $d = 1000, s = 1 \rightarrow a = 125000.166665$.
- Can use binary search over $a \in (0, 10^9]$.

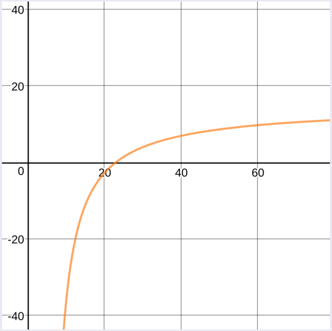


Figure : $f(a)$ for $s = 15$, $d = 50$

A – Birthday Cake – First solved at 49 minutes

Problem

Given n points on a circle and m lines cutting the circle, check if each face on the circle contains exactly one of the given points.

A – Birthday Cake – First solved at 49 minutes

Problem

Given n points on a circle and m lines cutting the circle, check if each face on the circle contains exactly one of the given points.

Solution

- Count the number of faces in the circle.
- Check if every pair of points are separated by at least one line.

A – Birthday Cake – First solved at 49 minutes

Solution Strategy

- The challenge is mainly about how to count the number of faces in the circle.
- Note that by the constraints, no two lines share a point on the circle boundary, and no three lines share a same point inside the circle.
- Thus, the number of faces inside the circle can be determined by counting the number of line-line intersections inside the circle (say s). The number of faces then equals $m + s + 1$, which you can verify with the following argument.
- Suppose we add the lines one by one. Each time we add a new line, it creates a new face as it enters the circle. With m lines we thus create m faces.
- A line also creates a new face when it hits one of the existing lines. As there are s intersections, there are s faces created this way. Add "1" to count the face of the original circle.

A – Birthday Cake – First solved at 49 minutes

Solution Strategy (cont'd)

- You can also prove this using the Euler Characteristic $F = 2 + E - V$.
- Side note: it is not necessary or useful to construct each of the faces to test whether it has a candle in it.

M – Umbral Decoding – First solved at 66 minutes

Problem

Given some points (p_i, q_i) each with a bound b_i,

find the number of points $(x, y) \in [0, N]^2$ that are not in any set:

$$\{(x, y) \mid (x - p_i)^3 + (y - q_i)^3 \leq b_i\}.$$

M – Umbral Decoding – First solved at 66 minutes

Problem

Given some points (p_i, q_i) each with a bound b_i,

find the number of points $(x, y) \in [0, N]^2$ that are not in any set:

$$\{(x, y) \mid (x - p_i)^3 + (y - q_i)^3 \leq b_i\}.$$

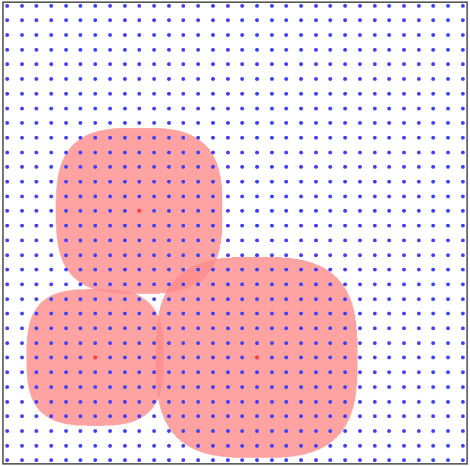
Solution

Use a quad-tree:
Divide the square into quadrants, and solve them recursively.

M – Umbral Decoding – First solved at 66 minutes

Quad Tree Strategy

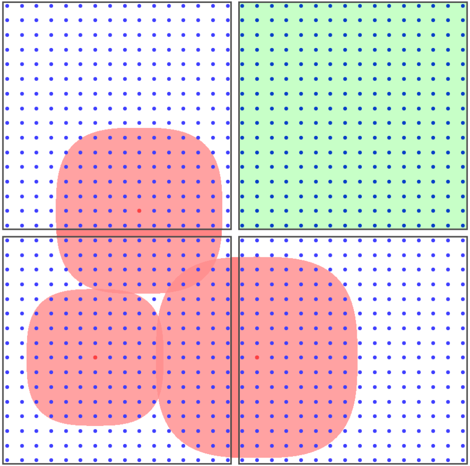
- Start with a single rectangle: (0, 0)-(N, N)



M – Umbral Decoding – First solved at 66 minutes

Quad Tree Strategy

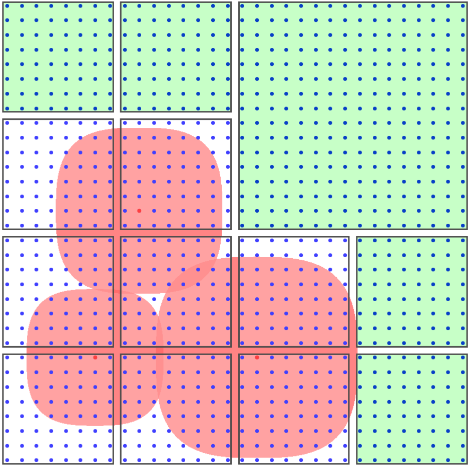
- Start with a single rectangle: (0, 0)-(N, N)
- Subdivide into quadrants



M – Umbral Decoding – First solved at 66 minutes

Quad Tree Strategy

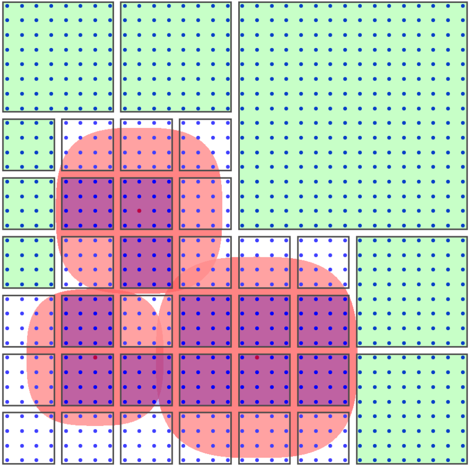
- Start with a single rectangle:
  (0, 0)-(N, N)
- Subdivide into quadrants
- If a quadrant lies entirely outside all umbras, count its points. This rectangle is done.
  (Green)



M – Umbral Decoding – First solved at 66 minutes

Quad Tree Strategy

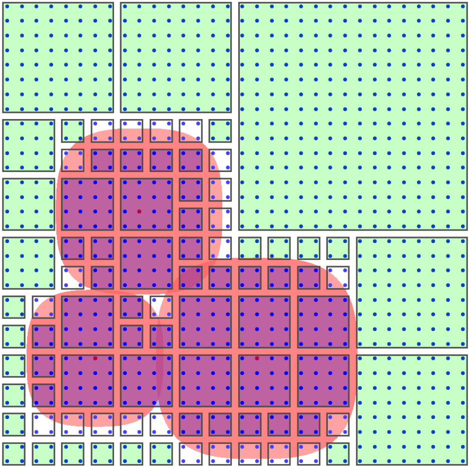
- If it lies entirely inside any *one* umbra, discard points. This rectangle is done.
  (Purple)



M – Umbral Decoding – First solved at 66 minutes

Quad Tree Strategy

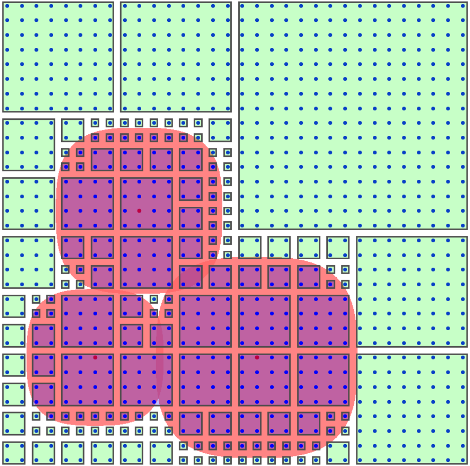
- If it lies entirely inside any *one* umbra, discard points. This rectangle is done. (Purple)
- The rest are processed recursively. (White)



M – Umbral Decoding – First solved at 66 minutes

Quad Tree Strategy

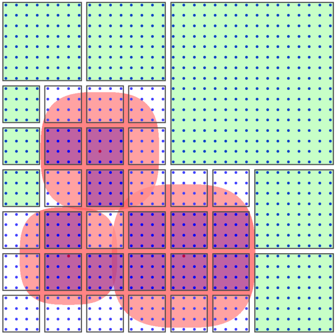
- If it lies entirely inside any *one* umbra, discard points. This rectangle is done. (Purple)
- The rest are processed recursively. (White)
- Some rectangles will contain a single point.



M – Umbral Decoding – First solved at 66 minutes

Analysis

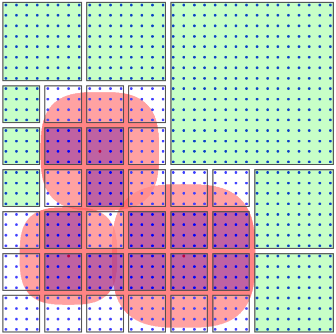
- This method can process very large regions in a single step.



M – Umbral Decoding – First solved at 66 minutes

Analysis

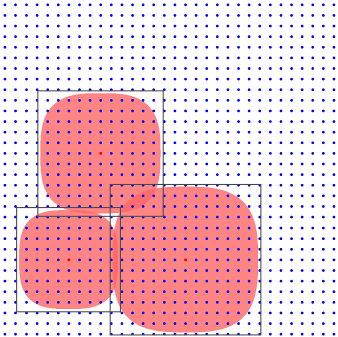
- This method can process very large regions in a single step.
- To optimize, associate with each rectangle a list of those safe points that might intersect it. This way, it can ignore the rest.



M – Umbral Decoding – First solved at 66 minutes

Simpler Solution

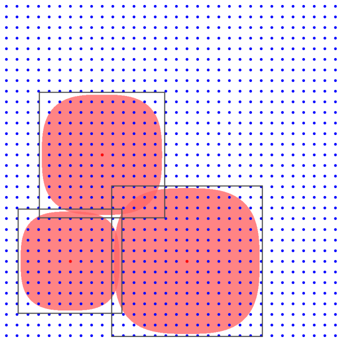
- Since $(10^8)^{1/3} < 465$, it is possible to simply count by brute force the points within a safe point's umbra, by searching a 930×930 box around the safe point.



M – Umbral Decoding – First solved at 66 minutes

Simpler Solution

- Since $(10^8)^{1/3} < 465$, it is possible to simply count by brute force the points within a safe point's umbra, by searching a 930×930 box around the safe point.
- Then save these to a set, sort, and count points different from their predecessors in the sorted list.



K – Space Probe – First solved at 78 minutes

Problem

- Given a set I of K pairwise disjoint intervals $[B_1, E_1], \ldots, [B_K, E_K]$,
- a strictly increasing sequence of N numbers $m_1, \ldots, m_N$, and
- additional interval $[T_1, T_2]$.

Choose a random value $T \in [T_1, T_2]$ (uniformly). What's the probability that the "shifted" sequence $T + m_1, \ldots, T + m_N$ doesn't intersect I; that is, $T + m_i \notin [B_j, E_j]$ for all $i = 1 \ldots N, j = 1 \ldots K$.

K – Space Probe – First solved at 78 minutes

Problem

- Given a set I of K pairwise disjoint intervals $[B_1, E_1], \dots, [B_K, E_K]$,
- a strictly increasing sequence of N numbers $m_1, \dots, m_N$, and
- additional interval $[T_1, T_2]$.

Choose a random value $T \in [T_1, T_2]$ (uniformly). What's the probability that the "shifted" sequence $T + m_1, \dots, T + m_N$ doesn't intersect I; that is, $T + m_i \notin [B_j, E_j]$ for all $i = 1 \dots N, j = 1 \dots K$.

Solution

For each $(m_i, [B_j, E_j])$ pair assume that they overlap and produce a forbidden range for the value T in $[T_1, T_2]$. Then sort those ranges, unify them, compute their total length L and output the value of $(|T_1 - T_2| - L)/|T_1 - T_2|$. Complexity: $O(KN \log(KN))$.

K – Space Probe – First solved at 78 minutes

Solution Strategy

- Create empty list *LFR* which stores each forbidden range as a pair $(start, end)$.
- Each pair of indices $(i, j), i = 1, \ldots, N, j = 1, \ldots, K$ may possibly produce one forbidden range. Rewrite the problem condition $T + m_i \notin [B_j, E_j]$ as $T \notin [B_j - m_i, E_j - m_i]$. The forbidden range $FR(i, j)$ associated with a particular pair of indices (i, j) is then $[max(B_j - m_i, T1), min(E_j - m_i, T2)]$.
- If the length of $FR(i, j)$ is positive append the pair $(max(B_j - m_i, T1), min(E_j - m_i, T2))$ to *LFR*.
- Update *LFR* for each $i = 1, \ldots N, j = 1, \ldots K$ and then sort *LFR* in ascending order according to the *start* component.

K – Space Probe – First solved at 78 minutes

Solution Strategy - Continued

- Introduce a *sumFreeLengths* variable which will measure the total length of "unforbidden" parts of $[T_1, T_2]$ and initialize it with 0.
- Introduce a *rightmostForbiddenEnd* variable which will register the current rightmost end of all forbidden ranges scanned so far. Initialize it with T_1.
- Scan LFR and for each $0 \leq k < length(LFR)$ do
  - increase *sumFreeLengths* by $max(0, LFR[k].start - rightmostForbiddenEnd)$,
  - update *rightmostForbiddenEnd* to $max(rightmostForbiddenEnd, LFR[k].end)$.
- Finally, increase *sumFreeLengths* by $T2 - rightmostForbiddenEnd$, if this difference is positive, and output $sumFreeLengths/|T_1 - T_2|$.

E – Company Picnic – First solved at 80 minutes

Problem

Given: a tree with a numeric value r_n at each node n.
Find: a maximum number of non-overlapping parent-child pairings that also maximizes the average of the smaller r_n value within each pairing.

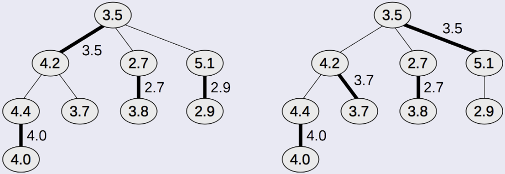


Figure : Pairing alternatives

E – Company Picnic – First solved at 80 minutes

Problem

Given: a tree with a numeric value r_n at each node n.
Find: a maximum number of non-overlapping parent-child pairings that also maximizes the average of the smaller r_n value within each pairing.

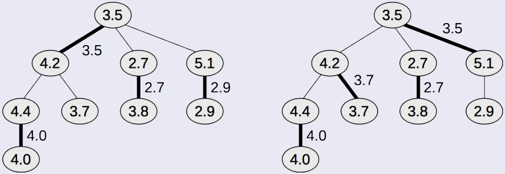


Figure : Pairing alternatives

Solution

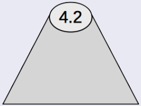
Dynamic programming (or memoization), computing the optimal solution in each subtree.

E – Company Picnic – First solved at 80 minutes

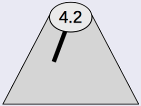
Optimal Value Structure

- A sub-problem for each node node n and boolean b

  $b = 0$ indicates n isn't paired with one of its (immediate) children, so it's available to pair with its parent.

  $b = 1$ indicates that n may be paired with a child, and is not available to pair with its parent.

- Define $v(n, b) = (p, s)$ as the optimal value for (n, b), where p is the number of pairings and s is the sum of the minimum running speeds for each pair in the subtree.
- Compute $v(n, b)$ for each node n and value of b using post-order DFS.

E – Company Picnic – First solved at 80 minutes

Computing $v(n, s)$ – in linear time

- Compute $v(n, b) = (p, s)$ for each node n and b using postorder DFS, with children values computed first.
- Let $c(n)$ be the immediate children of n.
- If node n is **not** paired with any of its children:

$$v(n, 0) = \sum_{i \in c(n)} v(i, 1)$$

- If node n **may be** paired with one of its children:

$$v(n, 1) = \max \begin{cases} v(n, 0) \\ \max_{i \in c(n)} v(n, 0) - v(i, 1) + v(i, 0) + (1, \min(r_n, r_i)) \end{cases}$$

- Base case, at a leaf, can't contain any pairs.
- Solution is $p, s/p$ for $(p, s) = v(root, 1)$.

D – Cat and Mice – First solved at 94 minutes

Problem

- Cartesian Cat hunts mice starting at time $t = 0$.
- Up to 15 mice are above ground for various lengths of time.
- To eat a mouse, the cat must get to it before it disappears.
- Eating a mouse slows the cat down by a constant multiplicative factor.
- What minimal starting velocity allows her to catch all of the mice?

D – Cat and Mice – First solved at 94 minutes

Problem

- Cartesian Cat hunts mice starting at time $t = 0$.
- Up to 15 mice are above ground for various lengths of time.
- To eat a mouse, the cat must get to it before it disappears.
- Eating a mouse slows the cat down by a constant multiplicative factor.
- What minimal starting velocity allows her to catch all of the mice?

Solution

- Brute-force strategy: try every permutation of visiting the mice, each time assuming unit starting velocity. After visiting all the mice, scale up the velocity to meet the needs of that permutation.
- Problem: with $n = 15$, the run time of $O(n \cdot n!)$ is too slow.
- Solution: use traveling salesman dynamic programming and binary search on the initial velocity.

D – Cat and Mice – First solved at 94 minutes

Solution Strategy

- Problem with dynamic programming approach
  - Need the ordering to calculate reachability.
  - Optimal solutions to larger problems might not be built from optimal solutions to smaller problems.
- What the DP can Solve
  - Given an initial velocity, we can calculate which subsets of mice are reachable.
  - We can also calculate the fastest time to eat those mice.
- Final Piece - Binary Search
  - Guess the necessary initial velocity and see if we can get all the mice.
  - If so, try something lower.
  - If not, try something higher.
  - Converge on the correct answer by halving the search space after each iteration.
  - The DP algorithm runs once for each iteration of the binary search.

References

[1] Robert Sedgewick and Kevin Wayne.
Algorithms and data structures.
https://www.cs.princeton.edu/~rs/AlgsDS07/15ShortestPaths.pdf.