ICPC Europe Regionals

icpc international collegiate programming contest

icpc.foundation

JET BRAINS
icpc global sponsor programming tools

HUAWEI
icpc diamond multi-regional sponsor

The 2021 ICPC Central Europe Regional Contest

ICPC CERC 2021

Solution presentation

Ljubljana, 24. 4. 2022

regionals 2021 icpc International Collegiate Programming Contest

F - Letters

Simulate shifting the letters in a matrix.

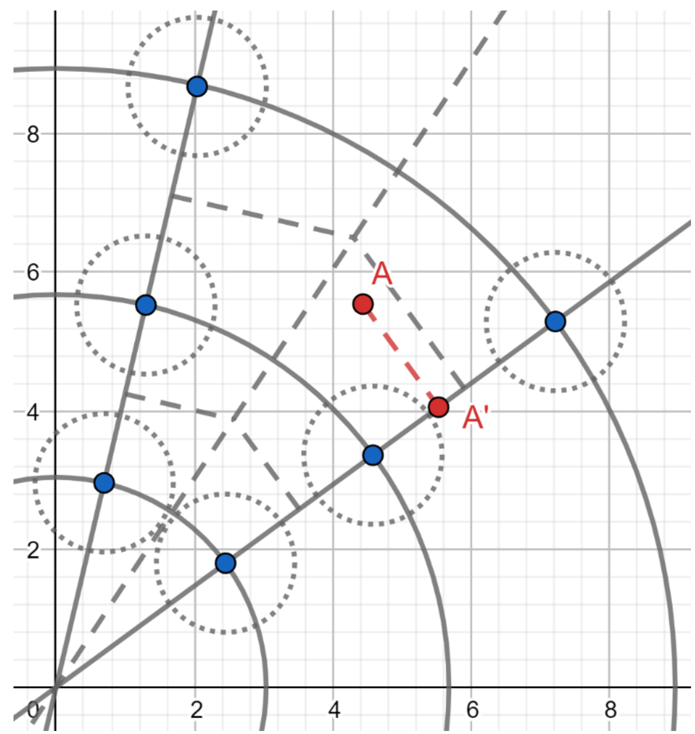
- low constraints (N, M, K <= 100)
- simulate all four directions (e.g., left)
  - process letters from left to right
  - shift each letter as far left as possible

```
k.l.ndi.            klndi...
.....c..            c.......
......ih    --->    ih......
j..a....            ja......
..cb....            cb......
..c...ef            cef.....
```

H - Radar

Find closest point from the intersection points of rays and concentric circles.

- too many intersections
- plane partitioning
  - binary search for circular sector
  - projection onto ray (A -> A’)
  - binary search for nearest point to A’

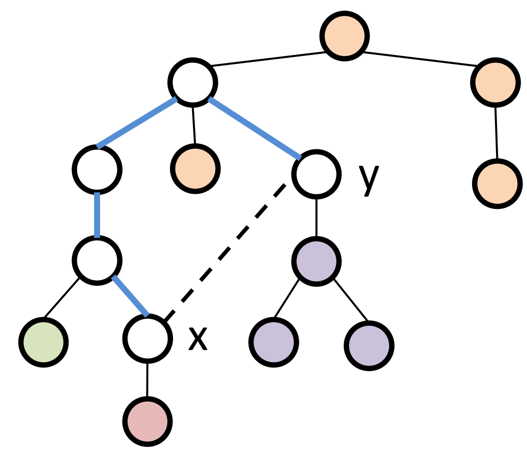


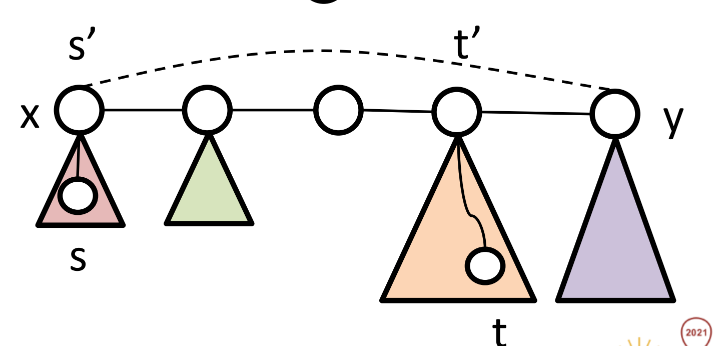
- careful: regions are not circular
- O(N (log R + log F))
- precision not an issue

A - Airline

Find the number of shortest paths affected by an addition of a new edge in a tree.

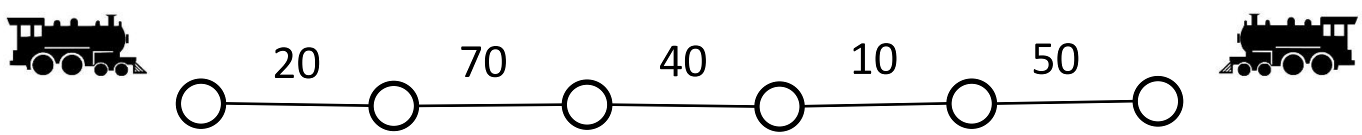
- find the path x – y
  - lowest common ancestor, two paths
  - small sum of d(x,y) … O(d), O(n log n)
- circular list of nodes with subtrees
  - d(s’, t’) > d(x,y)/2
  - compute size of subtrees
    - careful with LCA
- O(d)



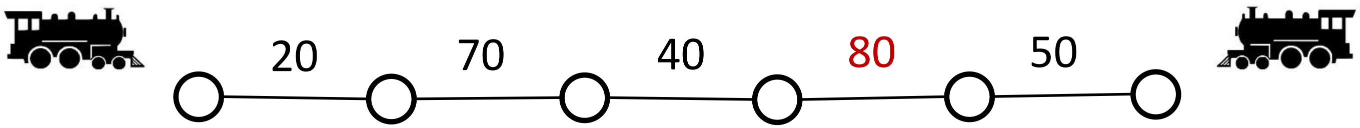


K - Single-track railway

Minimize waiting time for trains going in opposite directions along the same railway track.



- no updates
- assume left train must wait
  - it should move as far as possible
  - similar reasoning for the right one
- find the meeting point
  - prefix sums p_i, total time t
  - rightmost station such that p_i <= t/2, binary search
  - waiting time left(i) = (t - p_i) - p_i
  - answer = min(left(i), right(i+1))

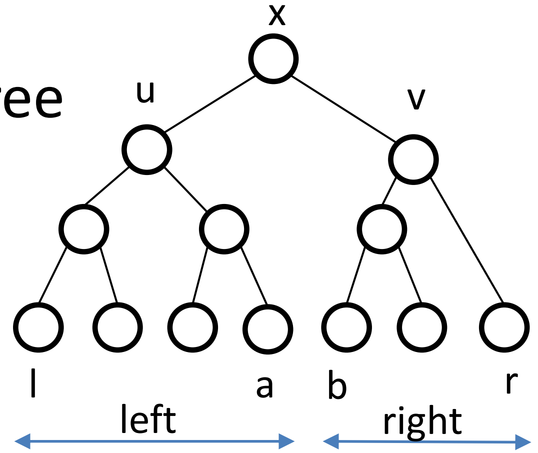


- updates?
  - data structure: modify value, compute prefix sum
  - Fenwick tree: log(n) update and prefix sum query
  - $O(k \log^2 n)$
  - Segment tree (static binary tree)
  - perform “binary search” by moving down the tree
  - $O(k \log n)$

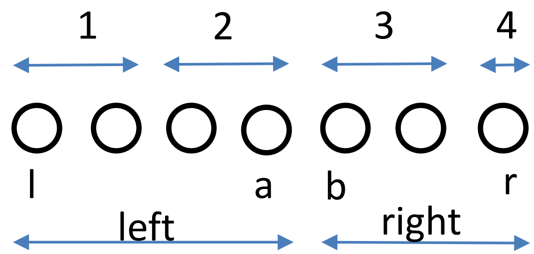
L - Systematic salesman

Find the optimal order of visiting left/right and top/bottom sets of points to minimize salesman's total path.

- represent recursive partitioning as a binary tree
  - sort, split, recurse
  - operation: swap left and right subtrees
  - goal: optimize leaf order



- $f(x, l, r) = \min_{a,b} f(u, l, a) + d(a, b) + f(v, b, r)$
  - min cost when l is the leftmost and r the rightmost in the subtree of x
  - $O(n^3)$ space? … x defined by l and r
  - pairs l and r, l and a, b and r should be from different subtrees
  - $f(r, l) = f(l, r)$
  - $O(n^4)$ dynamic programming is too slow



- $f(l, r) = \min_{a,b} f(l, a) + d(a, b) + f(b, r)$
  - $O(n^2)$ computation -> $O(n)$
  - split in two parts l – a – b and a – b – l
  - auxiliary function g (finds optimal a to get from l to b)
  - $g(l, b) = \min_a f(l, a) + d(a, b)$
  - $f(l, r) = \min_b g(l, b) + f(b, r)$

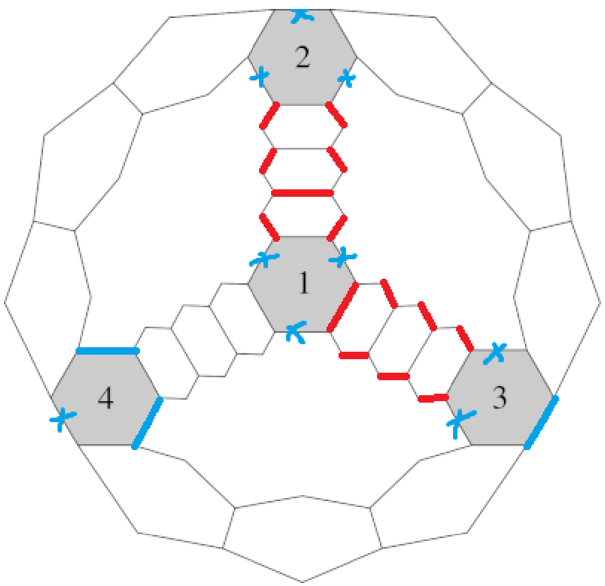
- time $O(n^3)$, space $O(n^2)$
- reconstruction: remember the optimal splits
- motivation: dendrograms (hierarchial clustering)
  - Bar-Joseph et al., Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics* (2001)
  - Bar-Joseph, Demaine et al. K-ary clustering with optimal leaf ordering for gene expression data. *Bioinformatics* (2003)

B - Building on the Moon

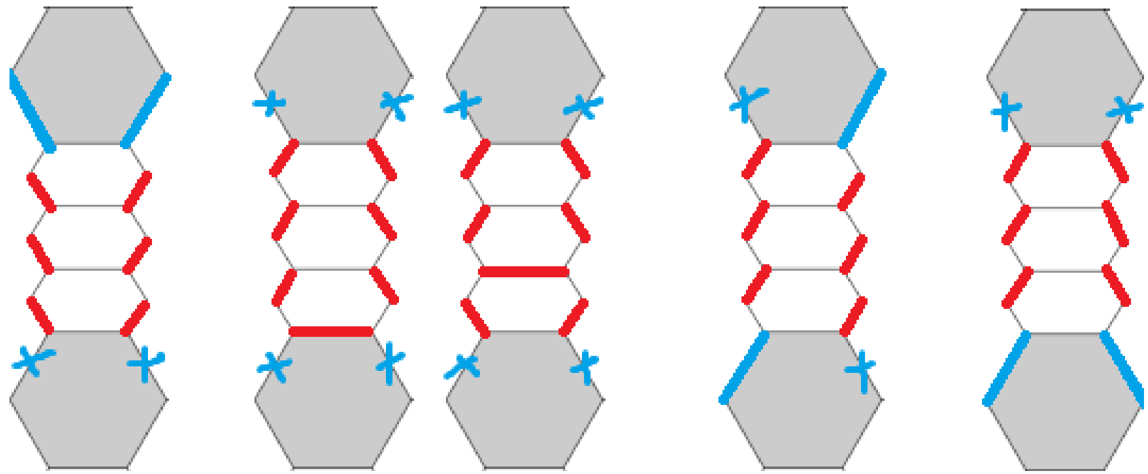
Count the number of maximum non-adjacent edge lightings in a hexagonal structure.

- count maximum matchings
- perfect matching (red)
  - use only passage edges

- small number of chambers (16)
- long passages (100)



- fix chamber edges that are not part of any passage (blue) and solve the "independent" passages



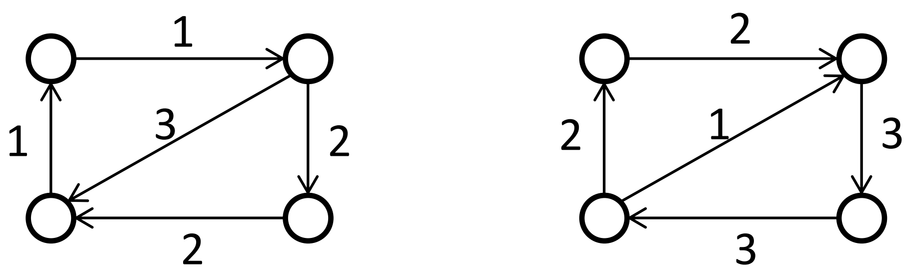
change: 1 unused: L+1 switch: 1 change: 1

- brute-force
  - 8^n infeasible (most cases don’t have a matching)
  - decide for adjacent chambers (e.g. in DFS/BFS order)
    - at most 4 cases, but mostly just 2
- additional improvements (not necessary)
  - fix the node with the currently lowest number of possible cases
  - dynamic programming with a profile of matched edges in outer chambers

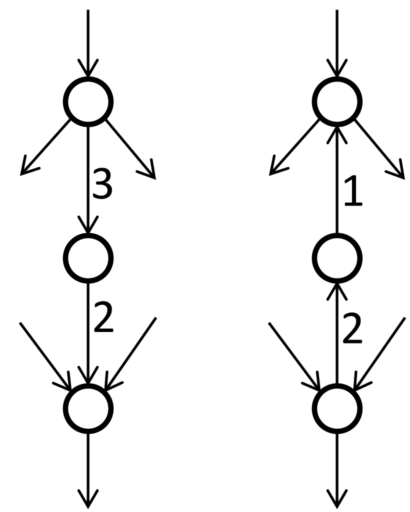
I - Regional development

Construct a nowhere-zero M-flow from a nowhere-zero flow modulo M.

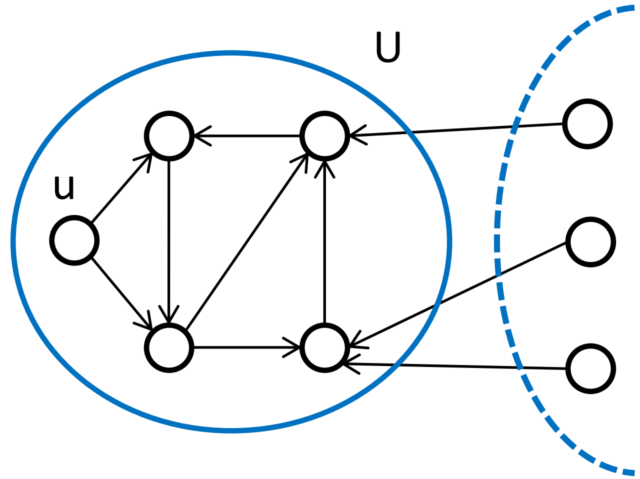
M=4



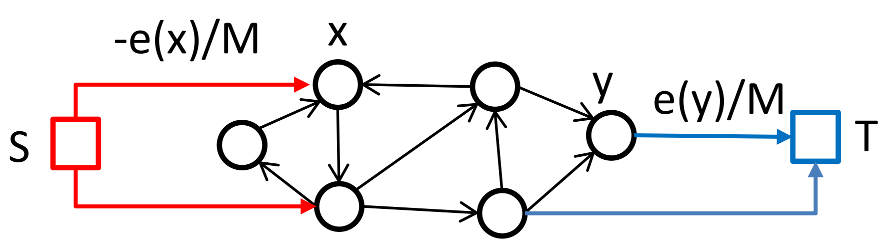
- modular flow violates Kirchhoff's law at nodes by k·M
- fix violations by "reversing" the flow
  - path from a deficit node to an excess node
  - send M units in the opposite direction
  - remains nowhere-zero
  - reduces violation at both ends by M



- solution always exists
  - e(x) = in(x) – out(x), $\sum e(x) = 0$
  - U … nodes reachable from u (e(u) < 0) in the direction of flow
    - all edges outside of U are incoming
  - there must exist a node v ∈ U with e(v) > 0 ($\sum_{x \in U} e(x) = in(U)$)



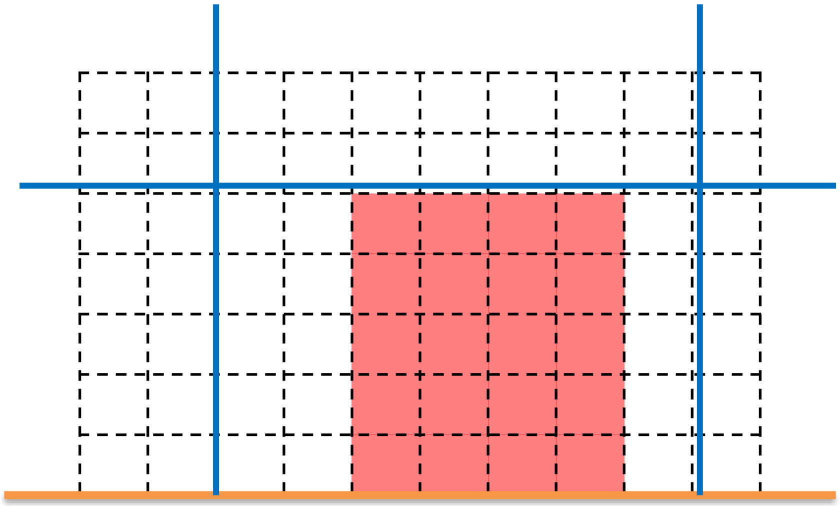
- O(R) violations by a factor of M … O(R (M + N))
- Ford-Fulkerson max flow
  - link up deficit and excess nodes along the positive edges



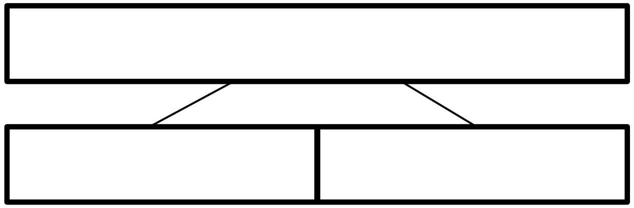
- nowhere-zero flows are related to colorings of planar graphs

E - Fishing

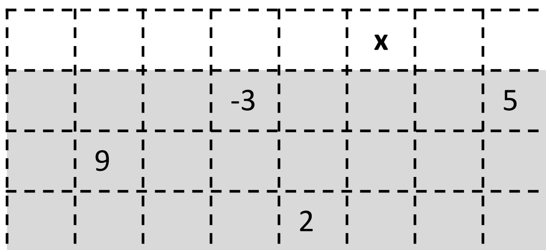
Find rectangles with maximum sum within given regions of a sparse matrix.

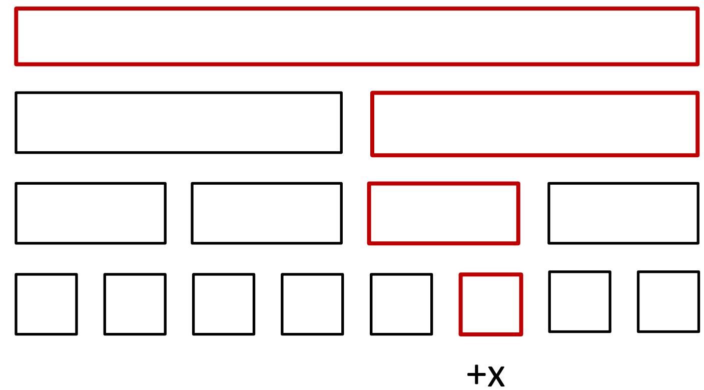


- fixed height, limited width
- $H_j = 1$
  - maximum subarray sum in range
  - segment tree
    - sum, max subarray sum, max prefix, max suffix
    - O(log M) query

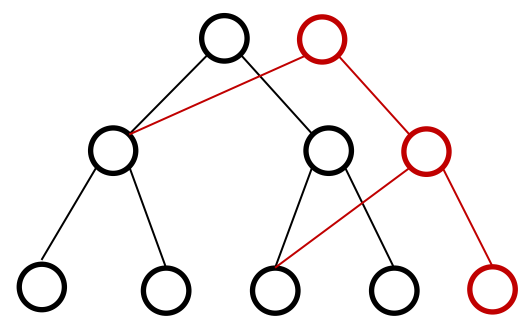


- H_j are increasing
  - matrix is sparse (only K nonzero entries)
  - update the segment tree with new elements (log M affected nodes)
  - O(K log M)





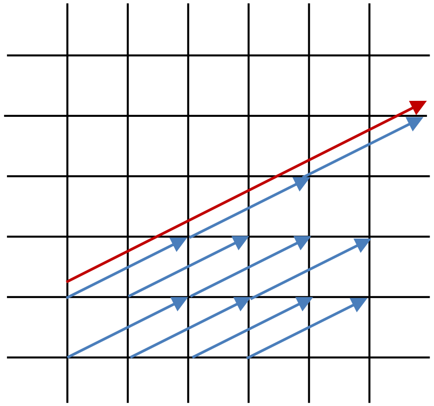
- solve for all heights and store segment trees
  - O(N M) … too large
  - build persistent segment trees (path copying)
  - precomputation, space: O(K log M)
  - query: O(log M)



G - Lines in a grid

Count the number of lines lying on at least two points of a grid.

- simplifications
  - #vert. = #horiz. = n
  - #incr. = #decr.
    - #flat = #steep, #diag. = 2n-3
- direction (d_x, d_y)
  - $1 <= d_y <= d_x <= n-1$, $gcd(d_x, d_y)=1$
  - $t(d_x, d_y) = (n - d_x)(n - d_y)$ offsets
  - $\#lines = \sum_{dx,dy} t(d_x, d_y) - t(2d_x, 2d_y) = f(n, 1) – f(n, 2)$
  - $f(n, k) = \sum_{dx,dy} (n - k\,d_x)(n - k\,d_y)$



- $f(n, k) = \sum_{dx=1..(n-1)/k} \sum_{dy=1..dx} (n - kd_x)(n - kd_y)$, $\quad gcd(d_x,d_y)=1$

$$= \sum_{dx=1..a} \sum_{dy=1..dx} (d_x \perp d_y)\ (n^2 - knd_x - knd_y + k^2d_xd_y)$$
$$= \sum_{dx=1..a} n^2\,\varphi(d_x) - kn\ d_x\ \varphi(d_x) - kn\ F(d_x) + k^2d_x\ F(d_x)$$
$$= \sum_{dx=1..a} n^2\,\varphi(d_x) - kn\ \varphi'(d_x) - kn\ F(d_x) + k^2F'(d_x)$$

  - precompute cumulative sums of φ, φ', F, F'
    - $\varphi(x)$ = number of integers coprime to x (Euler's phi)
      O(n log n)
    - F(x) = sum of integers coprime to x
      $F(x) = x\ \varphi(x)/2$ … numbers u and x-u coprime to x at the same time

D - DJ Darko

Update an array of speakers by increasing a range by x or by setting speakers in a range to a “normalized” value.

- ranges of speakers with the same value
  - store only differences (index, difference)
- increase volume
  - two changes: at the start (+x) and at the end (-x)
- get volume: prefix sum
- set volume
  - extract list of affected ranges and replace with a single range of normalized volume
- amortized analysis
  - increase introduces a const. number of new ranges
  - set removes some ranges (or adds at most two)

- computing the normalized volume v
  - sort by volume, “weighted” median

(volume: costs) … (2: 1+2+1), (5: 7+1), (1: 3+2+5), (7: 4), (2: 2+9)

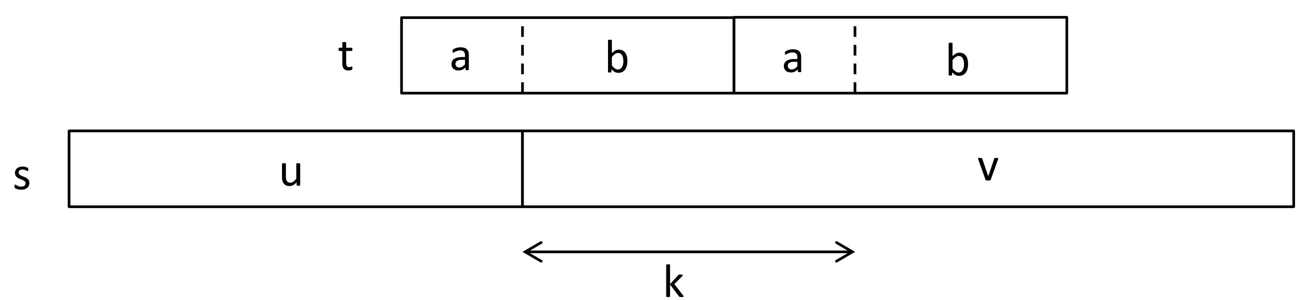
(1: 3+2+5), (2: 1+2+1), (2: 2+9), (5: 7+1), (7: 4)

  - one of existing volumes is optimal (or we could move it)
    - compare sum of costs in both directions (L and R)
  - move from i-th to (i+1)-th volume?
    - gain, loss per unit of volume … L_i + $cost_i$ < total_cost / 2
- O(q log n)
  - removing and adding ranges takes O(log n) per range
- practical considerations
  - introduce 0 differences to align ranges of speakers with the same volumes with query bounds
  - use a static tree (Fenwick, segment tree) and store locations of nonzero leaves in a separate set
    - find affected ranges in the set in O(log n)
    - find the actual volume of each range in O(log n)

J - Repetitions

Find the longest repetition for each given substring.

- square factors, Main-Lorentz (repetitions in a string)
  - divide & conquer: left half, right half, crossing the middle?
    - test(u,v) = testLeft(u,v), testRight(u,v)



  - consider different lengths k
  - b … pref(v, k) = longest prefix of v starting at k (z-algorithm)
  - a … suf(v, k) = longest suffix of u ending at k in v (pref(u’+v’))
  - |a| + |b| >= k, |a|,|b| <= k,   leftmost – maximize |a|

- complexity O(n log n)
  - O(|u|+|v|) for testing a pair of adjacent substrings
  - O(n) at each of the O(log n) levels

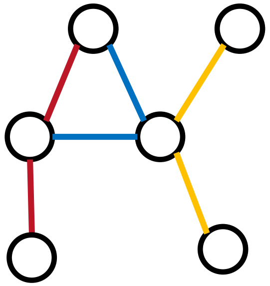
- generalize to substring queries
  - store results of the D&C tree
  - consider occurrences at bounds (test)
    - merge results from smaller towards larger sections of size 2^i

s_1	s_2	s_3	s_4	s_5

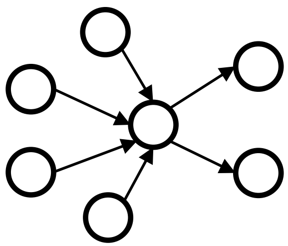
    - test(s_1, s_2), test(s_1+s_2, s_3)
    - test(s_4, s_5), test(s_3, s_4+s_5)
    - $|s_1|+|s_2| < 2|s_2|$, $|s_1|+|s_2|+|s_3| < 2|s_3|$
  - O(n)

C - Cactus Cutting

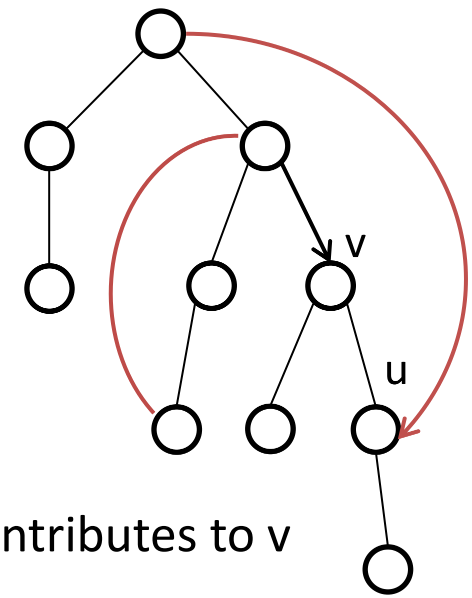
Count the number of ways of cutting a cactus graph into sticks (paths of length 2).



- directed sticks (towards center)
  - class of solutions
  - k incoming edges (even k)
    (k-1)(k-3)… = (k-1)!!
  - directing an edge produces two almost independent problems



- DFS tree
  - back-edges … disjoint cycles
  - f(v, p, c) … number of cuts into sticks of subtree rooted at v with parent edge pointing towards v (p=1) and cycle edge pointing down the tree (c=1)



  - every child edge can be directed either way and contributes to v
    - no edges: f(u, 1, c)
    - 1 edge: f(u, 0, c)
  - child whose edge is first in a cycle contributes:
    - no edges: f(u, 1, 1)
    - 1 edge: f(u, 0, 1) + f(u, 1, 0)
    - 2 edges: f(u, 0, 0)
  - find cases that contribute together exactly k edges?

- polynomials
  - each child represented as $(a+bx)$ or $(a+bx+cx^2)$
  - product: coefficient at x^k counts solutions that contribute k edges
    - consider contribution of p and c (in case of last node in a cycle)
  - multiply a list of polynomials (merge)
- FFT
  - careful with precision!
  - split the polynomial into two smaller ones $A(x) = A_1(x) + CA_2(x)$
    $A\,B = A_1\,B_1 + C\,(A_1\,B_2 + A_2\,B_1) + C^2\,(A_2\,B_2)$
  - 4 FFTs
- $O(n \log^2 n)$
- additional optimizations
  - multiply small polynomials naively
  - p has no effect on the product
  - c effects just one term in the product (child that is part of the cycle)
  - handle factors C or Cx separately (leaves)

The End