CTU Open 2024

Presentation of solutions

October 19, 2024

Flagbearer

Bear

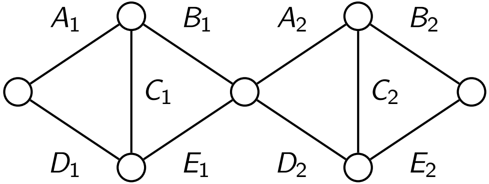
- Just rewrite the nice ascii art compare, shift and print.

Bear

- Just rewrite the nice ascii art compare, shift and print.
- You can also observe a regular pattern in the rotation of the hands (with few exceptions).

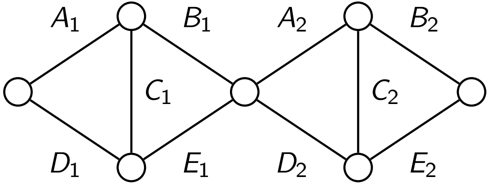
Fellow Sheep

Sheep



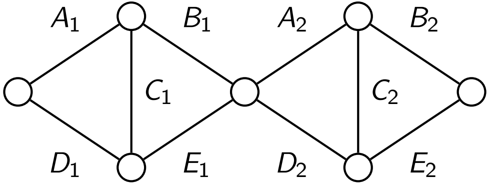
- The task boils down to finding the maximum flow between the pasture, and the farmyard.

Sheep



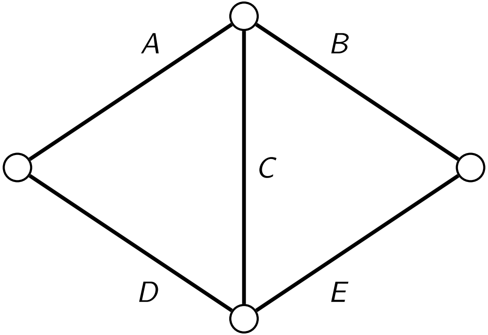
- The task boils down to finding the maximum flow between the pasture, and the farmyard.
- Maximum flow is equal to the minimum cut.

Sheep

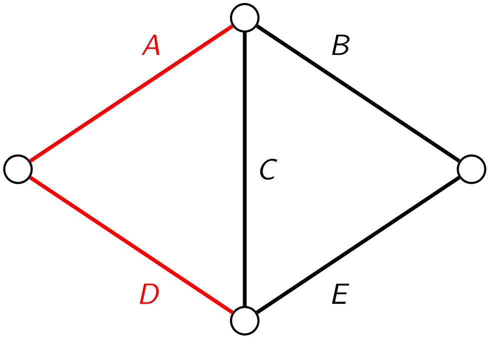


- The task boils down to finding the maximum flow between the pasture, and the farmyard.
- Maximum flow is equal to the minimum cut.
- Observation: for each segment, we can find the minimum cut separately.

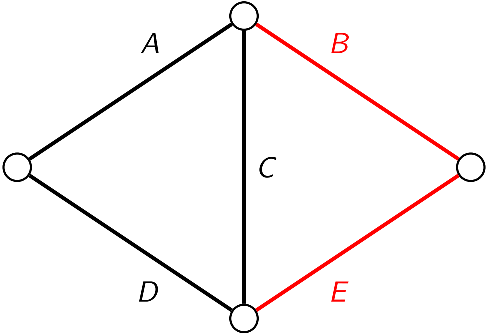
Sheep



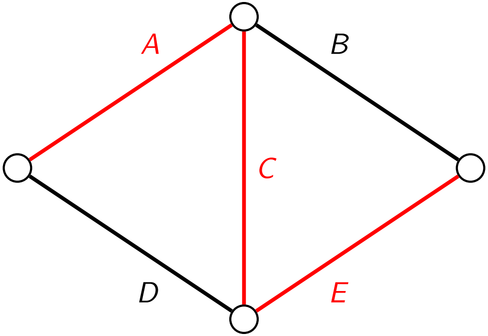
Sheep



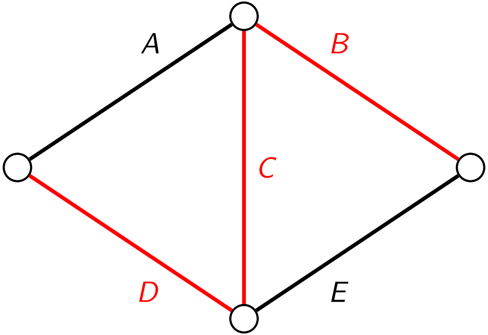
Sheep



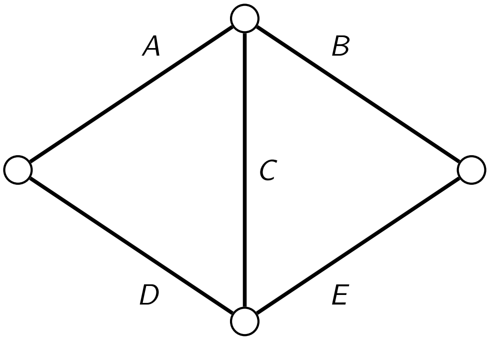
Sheep



Sheep



Sheep



$$min(A + D, B + E, A + C + E, B + C + D)$$

Sheep

Repeat for each segment, take the minimum of all segments.

Hamster

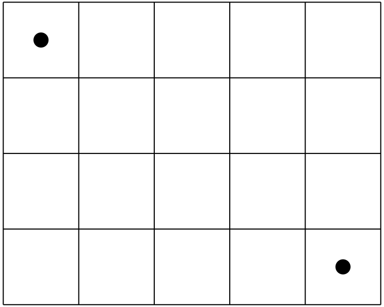
Hamster

- **Input:** $N \times M$ grid $P_N \times P_M$, each cell with nonegative integer.
- **Output:** Find a maximum cost path from top left to bottom right.

Hamster

- **Input:** $N \times M$ grid $P_N \times P_M$, each cell with nonegative integer.
- **Output:** Find a maximum cost path from top left to bottom right.

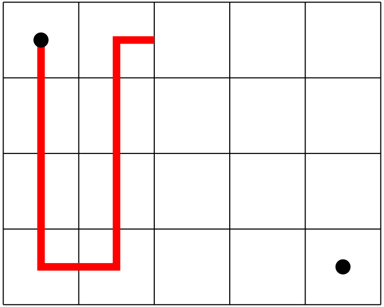
Observation 1: If N or M is odd there is a path going through all of the cells.



Hamster

- **Input:** $N \times M$ grid $P_N \times P_M$, each cell with nonegative integer.
- **Output:** Find a maximum cost path from top left to bottom right.

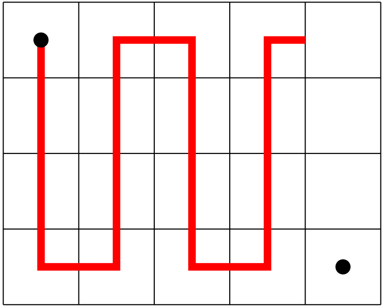
Observation 1: If N or M is odd there is a path going through all of the cells.



Hamster

- **Input:** $N \times M$ grid $P_N \times P_M$, each cell with nonegative integer.
- **Output:** Find a maximum cost path from top left to bottom right.

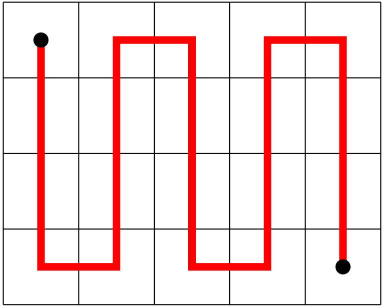
Observation 1: If N or M is odd there is a path going through all of the cells.



Hamster

- **Input:** $N \times M$ grid $P_N \times P_M$, each cell with nonegative integer.
- **Output:** Find a maximum cost path from top left to bottom right.

Observation 1: If N or M is odd there is a path going through all of the cells.



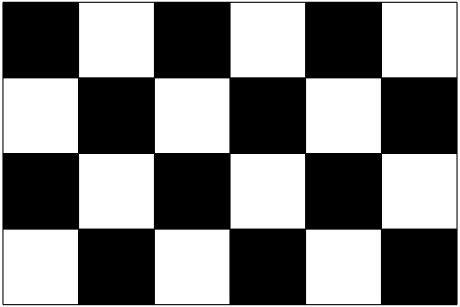
Hamster

So the interesting case is when both N and M are even.

Hamster

So the interesting case is when both N and M are even.

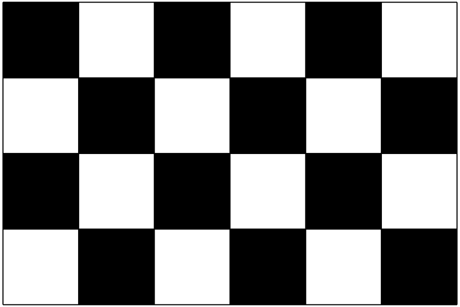
Observation 2: Color the grid with chessboard pattern, start and end cell are both black.



Hamster

So the interesting case is when both N and M are even.

Observation 2: Color the grid with chessboard pattern, start and end cell are both black.



Colorally: Each path contains one more black than white cell.

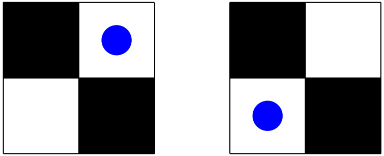
Hamster

Observation 3: For each white cell, there is a path that only removes that cell.

Hamster

Observation 3: For each white cell, there is a path that only removes that cell.

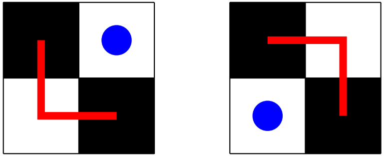
- **Base case:**



Hamster

Observation 3: For each white cell, there is a path that only removes that cell.

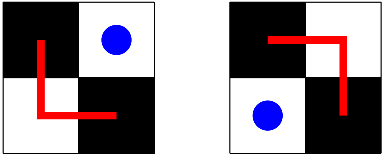
- **Base case:**



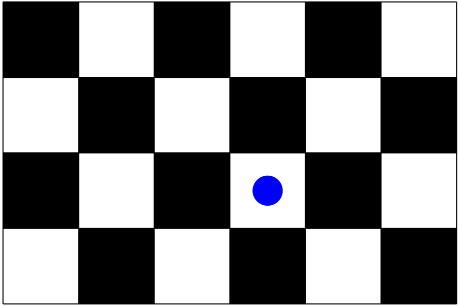
Hamster

Observation 3: For each white cell, there is a path that only removes that cell.

- **Base case:**



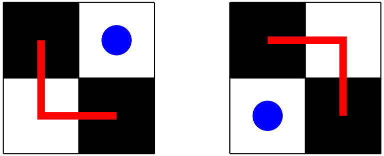
- **Induction step:** Remove 2 left/most most column or 2 first/last rows that do not contain the cell in question.



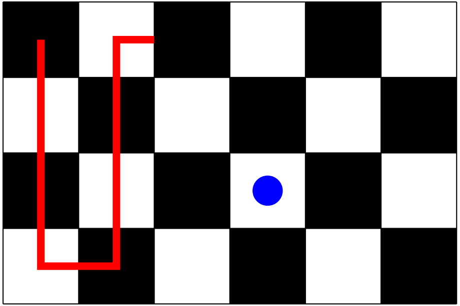
Hamster

Observation 3: For each white cell, there is a path that only removes that cell.

- **Base case:**



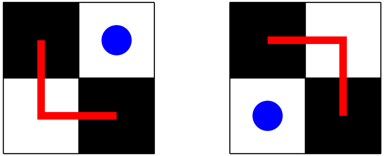
- **Induction step:** Remove 2 left/most most column or 2 first/last rows that do not contain the cell in question.



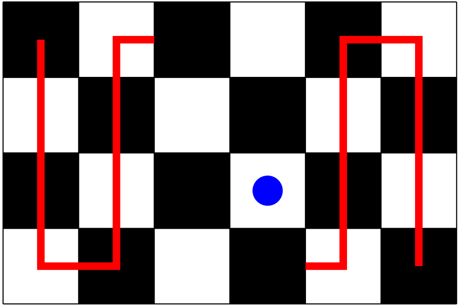
Hamster

Observation 3: For each white cell, there is a path that only removes that cell.

- **Base case:**



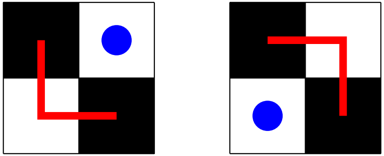
- **Induction step:** Remove 2 left/most most column or 2 first/last rows that do not contain the cell in question.



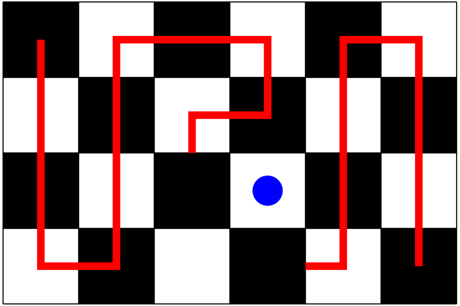
Hamster

Observation 3: For each white cell, there is a path that only removes that cell.

- **Base case:**



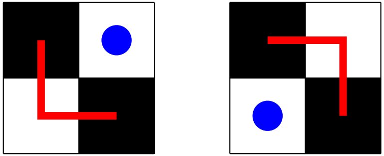
- **Induction step:** Remove 2 left/most most column or 2 first/last rows that do not contain the cell in question.



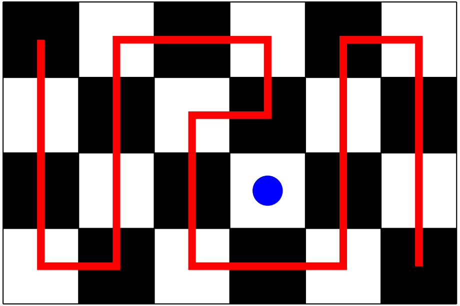
Hamster

Observation 3: For each white cell, there is a path that only removes that cell.

- **Base case:**



- **Induction step:** Remove 2 left/most most column or 2 first/last rows that do not contain the cell in question.



Hamster

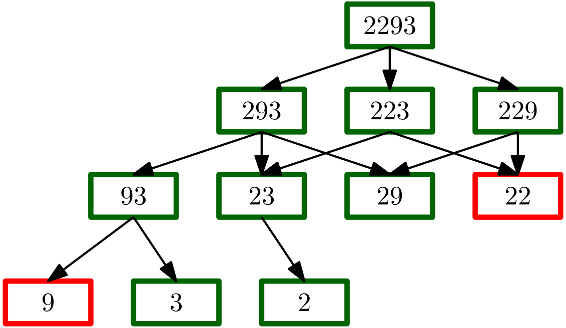
Summary:

- If N or M is odd, return sum of all cells.
- Else both N and M are even.
  - Let C_{MIN} be the minimum value on a cell that has sum of its coordinates odd.
  - Output sum of all cells minus C_{MIN}.

Pray mink

Mink

- **Input**: Integer $1 \leq N \leq 10^9$.
- **Task**: How many prime numbers can we obtain in a row when removing digits from N one by one?



Mink

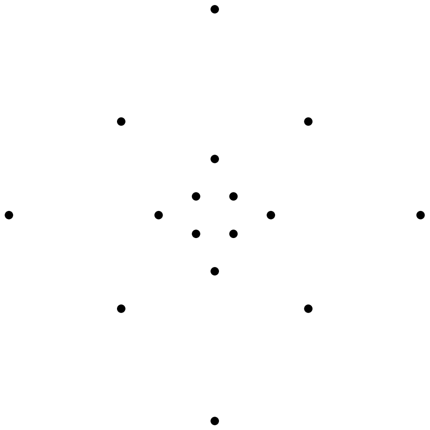
- N has at most 10 digits.
- We can obtain 2^{10} different numbers.
- We can recursively try all possible sequences of removing numbers and use DP.
- Checking primeness in time $O(\sqrt{n})$ is fast enough.
  - At most $\sum_{i=1}^{10} \binom{10}{i} \sqrt{10^i} \approx 1.6 \cdot 10^6$ modulo operations.
- The rest takes $O(2^{\log n}) = O(n)$ time.

$$ \tag{1} $$

Fishception

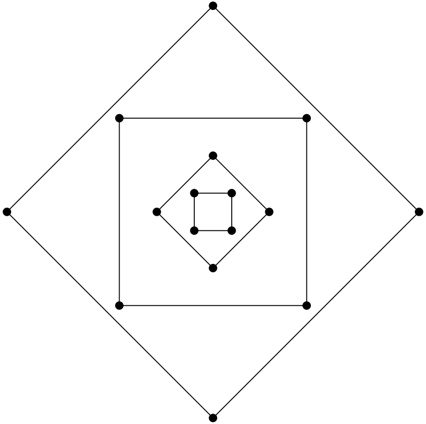
Fish

- $4N$ points with integer coordinates in a plane.
- The points represent the vertices of non-intersecting rectangles, each one contained within the next.
- **Task:** Determine the area of the smallest rectangle.



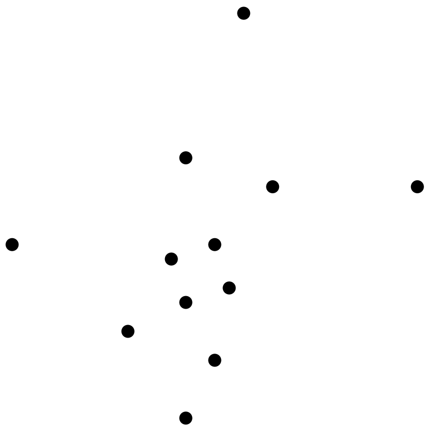
Fish

- $4N$ points with integer coordinates in a plane.
- The points represent the vertices of non-intersecting rectangles, each one contained within the next.
- **Task:** Determine the area of the smallest rectangle.



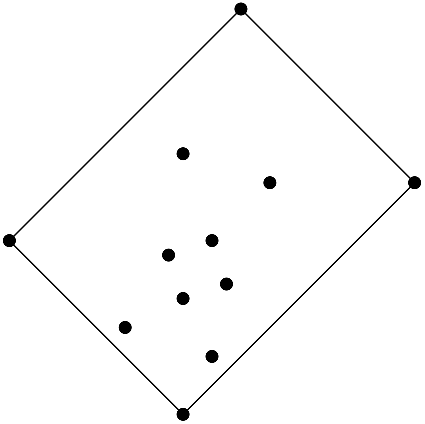
Fish

- Sort the points by the x-axis and by the y-axis.
- In each step, you take the leftmost, topmost, rightmost, and bottommost unmarked points and mark them.
- The last 4 points form the desired smallest rectangle.



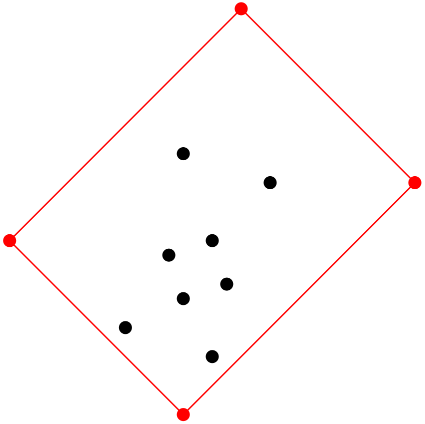
Fish

- Sort the points by the x-axis and by the y-axis.
- In each step, you take the leftmost, topmost, rightmost, and bottommost unmarked points and mark them.
- The last 4 points form the desired smallest rectangle.



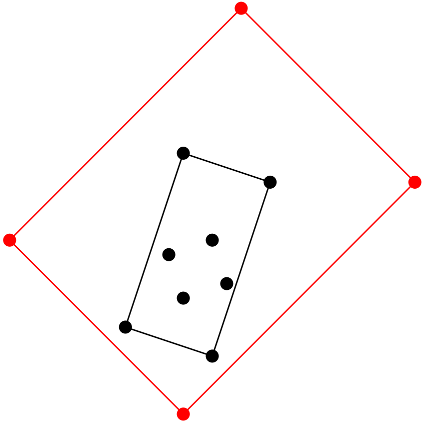
Fish

- Sort the points by the x-axis and by the y-axis.
- In each step, you take the leftmost, topmost, rightmost, and bottommost unmarked points and mark them.
- The last 4 points form the desired smallest rectangle.



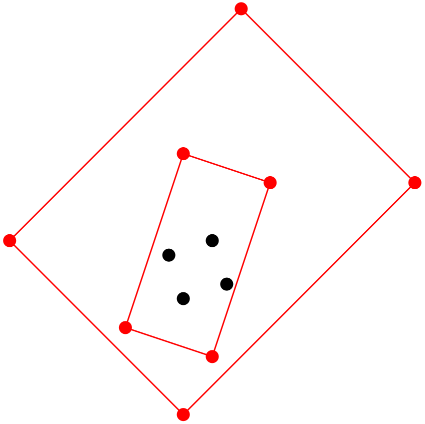
Fish

- Sort the points by the x-axis and by the y-axis.
- In each step, you take the leftmost, topmost, rightmost, and bottommost unmarked points and mark them.
- The last 4 points form the desired smallest rectangle.



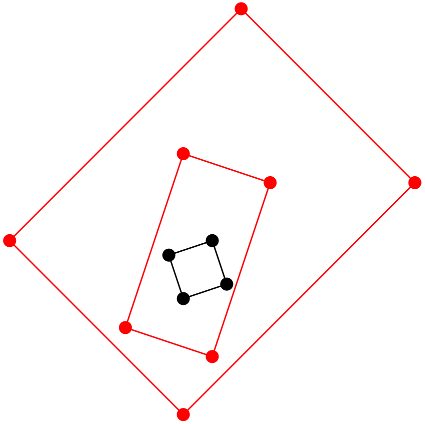
Fish

- Sort the points by the x-axis and by the y-axis.
- In each step, you take the leftmost, topmost, rightmost, and bottommost unmarked points and mark them.
- The last 4 points form the desired smallest rectangle.



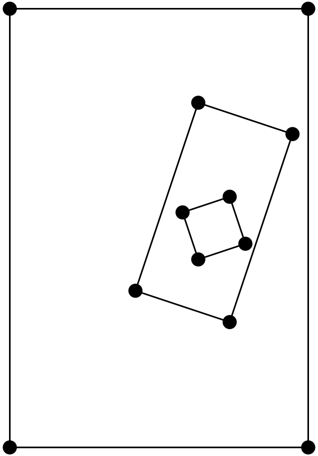
Fish

- Sort the points by the x-axis and by the y-axis.
- In each step, you take the leftmost, topmost, rightmost, and bottommost unmarked points and mark them.
- The last 4 points form the desired smallest rectangle.



Fish

- It is necessary to take care of the case when a rectangle has edges parallel to the axes.



Ornithology

Ornithology

- **Input**: A bipartite graph with two parts drawn on two parallel straight lines.
- **Output**: The number of edge crossings.
- The edges do not cross at the vertices.

Ornithology

Ornithology

- Purely combinatorial problem, no geometry involved.

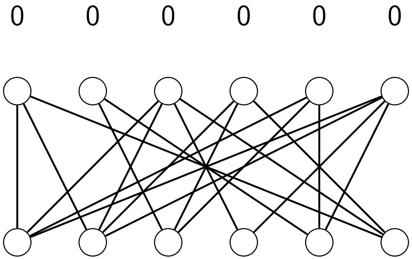
Ornithology

- Purely combinatorial problem, no geometry involved.
- Let $V = A \cup B$ and. Edges $\{a_1, b_1\}$ and $\{a_2, b_2\}$ cross if and only if $a_1 < a_2 \wedge b_2 < b_1$ or $a_1 > a_2 \wedge b_2 > b_1$.

Ornithology

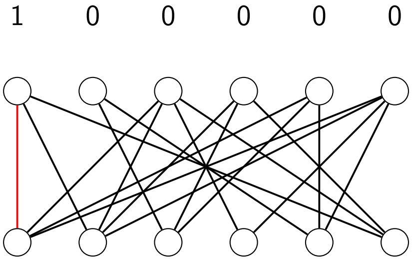
- Purely combinatorial problem, no geometry involved.
- Let $V = A \cup B$ and. Edges $\{a_1, b_1\}$ and $\{a_2, b_2\}$ cross if and only if $a_1 < a_2 \wedge b_2 < b_1$ or $a_1 > a_2 \wedge b_2 > b_1$.
- $O(|E|^2)$ naive algorithm is too slow. ($|E| \approx 2 \cdot 10^5$).

Ornithology - fast solution



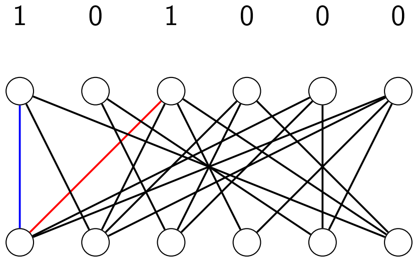
res =

Ornithology - fast solution



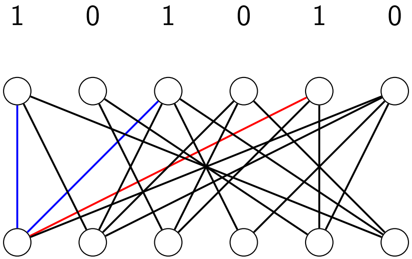
res =

Ornithology - fast solution



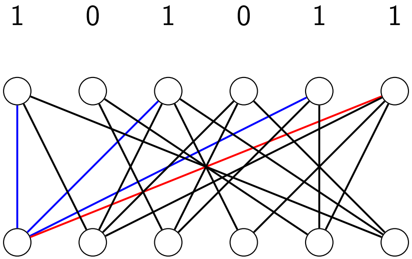
res =

Ornithology - fast solution



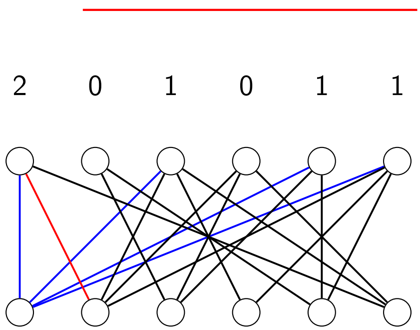
res =

Ornithology - fast solution



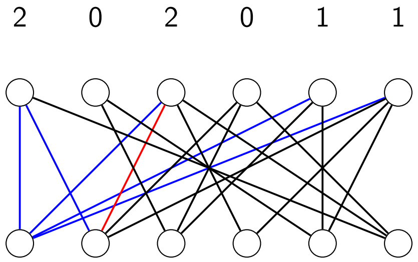
res =

Ornithology - fast solution



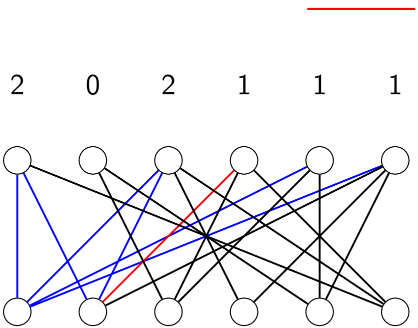
res = 3

Ornithology - fast solution



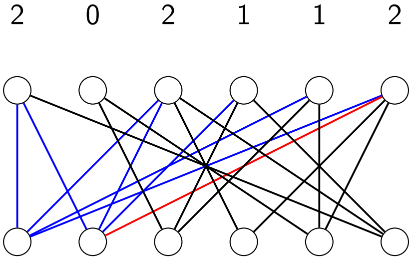
$$\text{res} = 3 + 2$$

Ornithology - fast solution



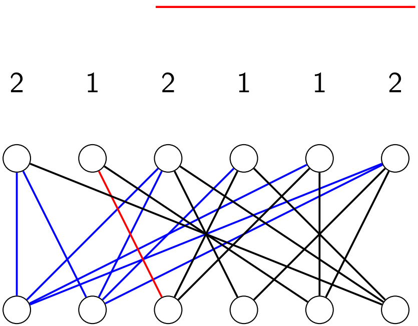
$$\text{res} = 3 + 2 + 2$$

Ornithology - fast solution



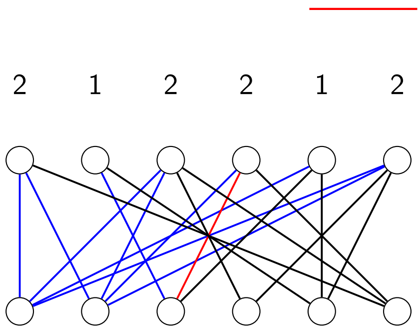
res $= 3 + 2 + 2$

Ornithology - fast solution



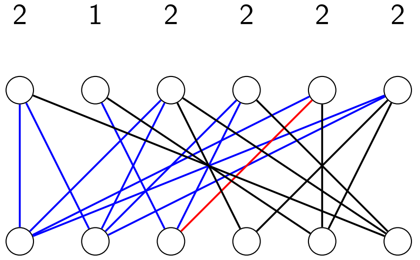
$$\text{res} = 3 + 2 + 2 + 6$$

Ornithology - fast solution



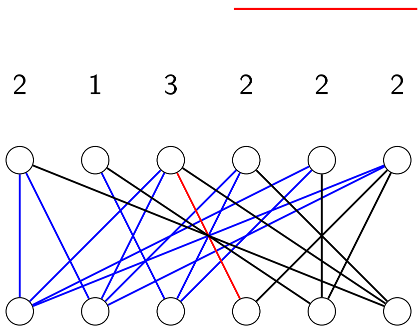
$$\text{res} = 3 + 2 + 2 + 6 + 3$$

Ornithology - fast solution



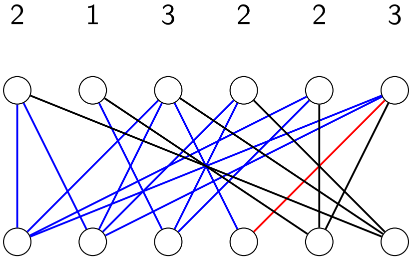
$\text{res} = 3 + 2 + 2 + 6 + 3 + 2$

Ornithology - fast solution



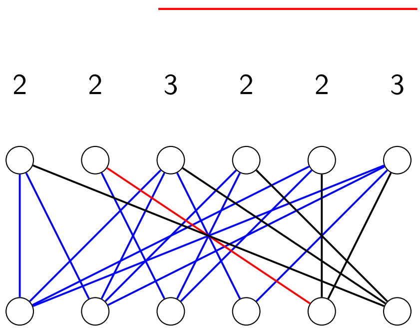
$$\text{res} = 3 + 2 + 2 + 6 + 3 + 2 + 6$$

Ornithology - fast solution



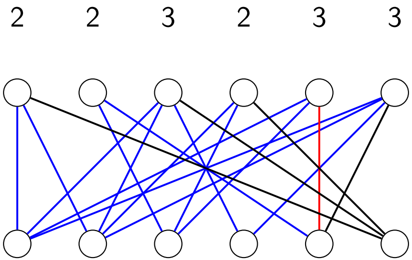
$$\text{res} = 3 + 2 + 2 + 6 + 3 + 2 + 6$$

Ornithology - fast solution



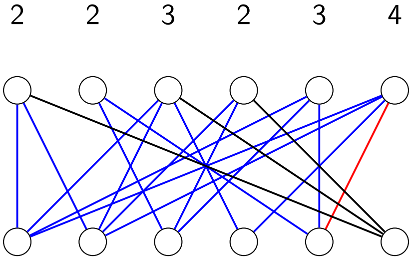
res $= 3 + 2 + 2 + 6 + 3 + 2 + 6 + 10$

Ornithology - fast solution



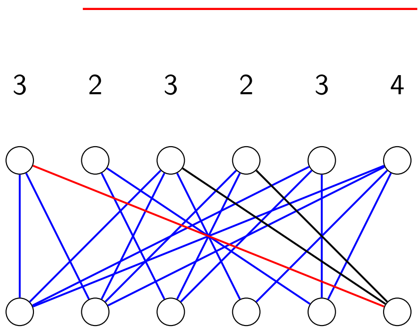
$$\text{res} = 3 + 2 + 2 + 6 + 3 + 2 + 6 + 10 + 3$$

Ornithology - fast solution



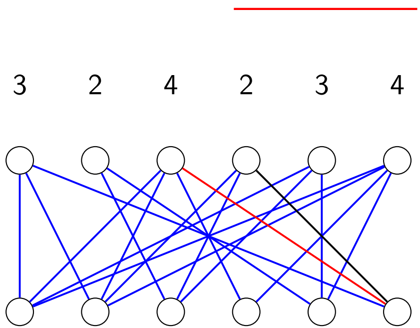
$$\text{res} = 3 + 2 + 2 + 6 + 3 + 2 + 6 + 10 + 3$$

Ornithology - fast solution



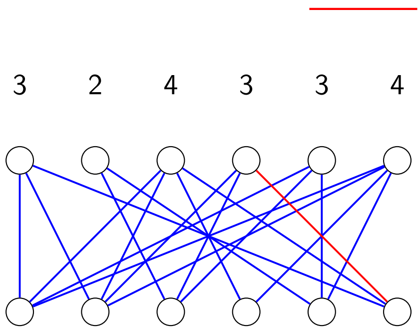
$$\text{res} = 3 + 2 + 2 + 6 + 3 + 2 + 6 + 10 + 3 + 14$$

Ornithology - fast solution



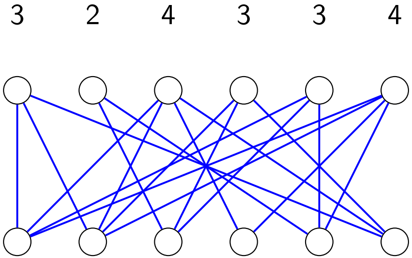
$$\text{res} = 3 + 2 + 2 + 6 + 3 + 2 + 6 + 10 + 3 + 14 + 9$$

Ornithology - fast solution



$$\text{res} = 3 + 2 + 2 + 6 + 3 + 2 + 6 + 10 + 3 + 14 + 9 + 7$$

Ornithology - fast solution



$$\text{res} = 3 + 2 + 2 + 6 + 3 + 2 + 6 + 10 + 3 + 14 + 9 + 7$$

Ornithology - fast solution

Ornithology - fast solution

- Sort edges lexicographically.

Ornithology - fast solution

- Sort edges lexicographically.
- Initialize an array x of size n with 0's

Ornithology - fast solution

- Sort edges lexicographically.
- Initialize an array x of size n with 0's
- For edge $\{a, b\}$ add 1 to index b and add $\sum_{i=b+1}^{n-1} x_i$ to the result.

Ornithology - fast solution

- Sort edges lexicographically.
- Initialize ~~an array~~ suitable data structure x of size n with 0's
- For edge $\{a, b\}$ add 1 to index b and add $\sum_{i=b+1}^{n-1} x_i$ to the result.
- Actually instead of a plain array, use your favourite data structure with fast point update and range sum query.

Ornithology - fast solution

- Sort edges lexicographically.
- Initialize ~~an array~~ suitable data structure x of size n with 0's
- For edge $\{a, b\}$ add 1 to index b and add $\sum_{i=b+1}^{n-1} x_i$ to the result.
- Actually instead of a plain array, use your favourite data structure with fast point update and range sum query.
- Anything reasonable with $o(N)$ complexity per query was intended to pass (segment tree, fenwick tree, sqrt decomposition,...).

Ornithology - fast solution

- Sort edges lexicographically.
- Initialize ~~an array~~ suitable data structure x of size n with 0's
- For edge $\{a, b\}$ add 1 to index b and add $\sum_{i=b+1}^{n-1} x_i$ to the result.
- Actually instead of a plain array, use your favourite data structure with fast point update and range sum query.
- Anything reasonable with $o(N)$ complexity per query was intended to pass (segment tree, fenwick tree, sqrt decomposition,...).
- Complexity: $O(|E| \log N)$ with e.g. segment tree.

Ornithology - fast solution without data structures

Ornithology - fast solution without data structures

- Divide and Conquer approach.

Ornithology - fast solution without data structures

- Divide and Conquer approach.
- Rough Idea: Split edges to two parts based on the endpoint in the first part and recurse.

Ornithology - fast solution without data structures

- Divide and Conquer approach.
- Rough Idea: Split edges to two parts based on the endpoint in the first part and recurse.
- Merge step: calculate the intersections between edges that have one endpoint in left and one in right.

Ornithology - fast solution without data structures

- Divide and Conquer approach.
- Rough Idea: Split edges to two parts based on the endpoint in the first part and recurse.
- Merge step: calculate the intersections between edges that have one endpoint in left and one in right.
- Complexity $O(|E| \log N)$.

Pork cutting

Pork

- Decompose each number into bit.
- We need have to find such intervals, which have the number of 0-bits of K equal to 0 while the number of 1-bits of K to be at least 1.

Pork

- Decompose each number into bit.
- We need have to find such intervals, which have the number of 0-bits of K equal to 0 while the number of 1-bits of K to be at least 1.
- Sweep the array and calculate prefix sum for each bit.
- Keep the track of "correct" intervals by two pointers.

Pork

- Decompose each number into bit.
- We need have to find such intervals, which have the number of 0-bits of K equal to 0 while the number of 1-bits of K to be at least 1.
- Sweep the array and calculate prefix sum for each bit.
- Keep the track of "correct" intervals by two pointers.
- Complexity: $O(N\dot{l}og(K))$

Rabid rabbit

Rabbit

- Observe that there is relatively small number of Fibonacci numbers - less than $(log(|U|))$

Rabbit

- Observe that there is relatively small number of Fibonacci numbers - less than $(log(|U|))$
- Solve the problem for each Fibonacci number separately.

Rabbit

- Observe that there is relatively small number of Fibonacci numbers - less than $(log(|U|))$
- Solve the problem for each Fibonacci number separately.
- Iterate through array, keeping track of last occurence of every number.
- Use two pointers technique to find "least interval" for each beginning, where the actual Fibonacci number can be constructed.

Rabbit

- Observe that there is relatively small number of Fibonacci numbers - less than $(log(|U|))$
- Solve the problem for each Fibonacci number separately.
- Iterate through array, keeping track of last occurence of every number.
- Use two pointers technique to find "least interval" for each beginning, where the actual Fibonacci number can be constructed.
- Complexity: $O(N\dot{l}og(|U|)\dot{l}og(|U|) + Q\dot{l}og(|U|))$

Watchdogs

Watch~~dogs~~

Watchcats

Watchcats

- **Input:** Tree T with q paths specified by endpoints.
- **Task:** For each a-b path P a vertex x on P with $|d(a,x) - d(b,x)| \leq 1$ must be selected. One vertex can be selected for multiple paths.
- **Output:** Minimum number of vertices to select.

Watchcat Niki

Watchcat Míša

Watchcats

- Each a-b path (for pair of lairs (a, b)) has some set of vertices corresponding to vulnerability places – the *vulnerability set*.

Watchcats

- Each a-b path (for pair of lairs (a, b)) has some set of vertices corresponding to vulnerability places – the *vulnerability set*.
- Each vulnerability set is either a single vertex or induces an edge.

Watchcats

- Each a-b path (for pair of lairs (a, b)) has some set of vertices corresponding to vulnerability places – the *vulnerability set*.
- Each vulnerability set is either a single vertex or induces an edge.
- If it is a vertex, it must be covered by a watchcat. Let $S \subseteq V$ be the set of vertices v s.t.$\{v\}$ is the vulnerability set for some mouse.

Watchcats

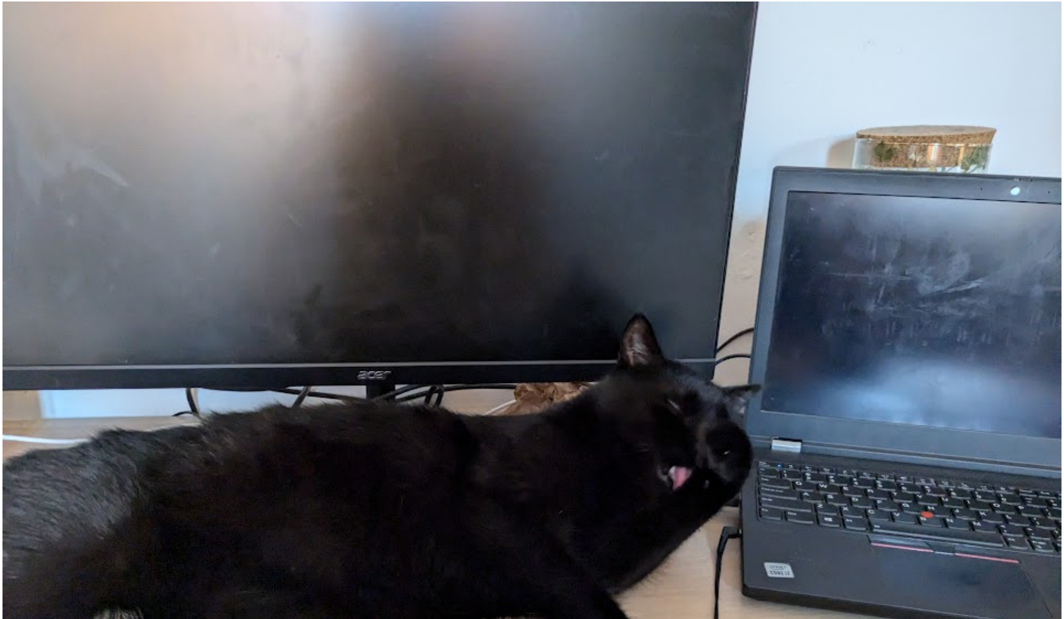
- Each a-b path (for pair of lairs (a, b)) has some set of vertices corresponding to vulnerability places – the *vulnerability set*.
- Each vulnerability set is either a single vertex or induces an edge.
- If it is a vertex, it must be covered by a watchcat. Let $S \subseteq V$ be the set of vertices v s.t.$\{v\}$ is the vulnerability set for some mouse.
- In the graph $T[V(T) \setminus S]$ we are left with edges that must be covered.

Watchcats

- Each a-b path (for pair of lairs (a, b)) has some set of vertices corresponding to vulnerability places – the *vulnerability set*.
- Each vulnerability set is either a single vertex or induces an edge.
- If it is a vertex, it must be covered by a watchcat. Let $S \subseteq V$ be the set of vertices v s.t.$\{v\}$ is the vulnerability set for some mouse.
- In the graph $T[V(T) \setminus S]$ we are left with edges that must be covered.
- Vertex Cover on the remaining forest!

Watchcat Čičinas

Watchcat Denis



Watchcats - How to solve it fast?

- The vertex cover can be solved in $O(n)$ time on trees by DP:

Watchcats - How to solve it fast?

- The vertex cover can be solved in $O(n)$ time on trees by DP:
- Hang the tree on r and for each vertex in bottom up manner compute:

Watchcats - How to solve it fast?

- The vertex cover can be solved in $O(n)$ time on trees by DP:
- Hang the tree on r and for each vertex in bottom up manner compute:
- $D[v][0]$ - the size of smallest vertex cover S in the subtree rooted at v where $v \notin S$.

Watchcats - How to solve it fast?

- The vertex cover can be solved in $O(n)$ time on trees by DP:
- Hang the tree on r and for each vertex in bottom up manner compute:
- $D[v][0]$ - the size of smallest vertex cover S in the subtree rooted at v where $v \notin S$.
- $D[v][1]$ - the size of smallest vertex cover S in the subtree rooted at v where $v \in S$.

Watchcats - How to solve it fast?

- The vertex cover can be solved in $O(n)$ time on trees by DP:
- Hang the tree on r and for each vertex in bottom up manner compute:
- $D[v][0]$ - the size of smallest vertex cover S in the subtree rooted at v where $v \notin S$.
- $D[v][1]$ - the size of smallest vertex cover S in the subtree rooted at v where $v \in S$.
- Transition: $D[v][0] = \sum_{u \in \mathrm{child}(v)} D[u][1]$ and $D[v][1] = \sum_{u \in \mathrm{child}(v)} \min\{D[u][0], D[u][1]\}$.

Watchcats - How to solve it fast?

- The vertex cover can be solved in $O(n)$ time on trees by DP:
- Hang the tree on r and for each vertex in bottom up manner compute:
- $D[v][0]$ - the size of smallest vertex cover S in the subtree rooted at v where $v \notin S$.
- $D[v][1]$ - the size of smallest vertex cover S in the subtree rooted at v where $v \in S$.
- Transition: $D[v][0] = \sum_{u \in \text{child}(v)} D[u][1]$ and $D[v][1] = \sum_{u \in \text{child}(v)} \min\{D[u][0], D[u][1]\}$.
- Result is $\min\{D[r][0], D[r][1]\}$.

Watchcat Ritchie

Watchcat Kiwi

Watchcats - what about the vulnerability sets?

- Naively in $O(n)$ per each mouse. Worst case $\Omega(nq)$ – too slow.

Watchcats - what about the vulnerability sets?

- Naively in $O(n)$ per each mouse. Worst case $\Omega(nq)$ – too slow.
- Build LCA!

Watchcats - what about the vulnerability sets?

- Naively in $O(n)$ per each mouse. Worst case $\Omega(nq)$ – too slow.
- Build LCA! Suppose that a, b is a pair of lairs for some mouse and let $\ell = \mathrm{lca}(a, b)$.

Watchcats - what about the vulnerability sets?

- Naively in $O(n)$ per each mouse. Worst case $\Omega(nq)$ – too slow.
- Build LCA! Suppose that a, b is a pair of lairs for some mouse and let $\ell = \mathrm{lca}(a, b)$.
- If $d(\ell, a) = d(\ell, b)$, then the resulting set is just $\{\ell\}$.

Watchcats - what about the vulnerability sets?

- Naively in $O(n)$ per each mouse. Worst case $\Omega(nq)$ – too slow.
- Build LCA! Suppose that a, b is a pair of lairs for some mouse and let $\ell = \text{lca}(a, b)$.
- If $d(\ell, a) = d(\ell, b)$, then the resulting set is just $\{\ell\}$.
- If $d(\ell, a) < d(\ell, b)$, then the result lies on the path from a to ℓ.

Watchcats - what about the vulnerability sets?

- Naively in $O(n)$ per each mouse. Worst case $\Omega(nq)$ – too slow.
- Build LCA! Suppose that a, b is a pair of lairs for some mouse and let $\ell = \mathrm{lca}(a, b)$.
- If $d(\ell, a) = d(\ell, b)$, then the resulting set is just $\{\ell\}$.
- If $d(\ell, a) < d(\ell, b)$, then the result lies on the path from a to ℓ.
- To find the central vertices of the path, jump from a to ℓ to distance $t = \lfloor d(\ell, a)/2 \rfloor$ in $O(\log n)$ steps using precomputed jumps from LCA. Let x be the vertex at distance t from a.

Watchcats - what about the vulnerability sets?

- Naively in $O(n)$ per each mouse. Worst case $\Omega(nq)$ – too slow.
- Build LCA! Suppose that a, b is a pair of lairs for some mouse and let $\ell = \text{lca}(a, b)$.
- If $d(\ell, a) = d(\ell, b)$, then the resulting set is just $\{\ell\}$.
- If $d(\ell, a) < d(\ell, b)$, then the result lies on the path from a to ℓ.
- To find the central vertices of the path, jump from a to ℓ to distance $t = \lfloor d(\ell, a)/2 \rfloor$ in $O(\log n)$ steps using precomputed jumps from LCA. Let x be the vertex at distance t from a.
- Distinguish the case when the vulnerability set contains one or two vertices based on parity of $d(a, b)$. Even – $\{x\}$, odd – $\{x, p(x)\}$ ($p(x)$ is the parent of x).

Watchcats - what about the vulnerability sets?

- Naively in $O(n)$ per each mouse. Worst case $\Omega(nq)$ – too slow.
- Build LCA! Suppose that a, b is a pair of lairs for some mouse and let $\ell = \text{lca}(a, b)$.
- If $d(\ell, a) = d(\ell, b)$, then the resulting set is just $\{\ell\}$.
- If $d(\ell, a) < d(\ell, b)$, then the result lies on the path from a to ℓ.
- To find the central vertices of the path, jump from a to ℓ to distance $t = \lfloor d(\ell, a)/2 \rfloor$ in $O(\log n)$ steps using precomputed jumps from LCA. Let x be the vertex at distance t from a.
- Distinguish the case when the vulnerability set contains one or two vertices based on parity of $d(a, b)$. Even – $\{x\}$, odd – $\{x, p(x)\}$ ($p(x)$ is the parent of x).
- The case $d(\ell, a) > d(\ell, b)$ is symmetric.

Watchcats - what about the vulnerability sets?

- Naively in $O(n)$ per each mouse. Worst case $\Omega(nq)$ – too slow.
- Build LCA! Suppose that a, b is a pair of lairs for some mouse and let $\ell = \mathrm{lca}(a, b)$.
- If $d(\ell, a) = d(\ell, b)$, then the resulting set is just $\{\ell\}$.
- If $d(\ell, a) < d(\ell, b)$, then the result lies on the path from a to ℓ.
- To find the central vertices of the path, jump from a to ℓ to distance $t = \lfloor d(\ell, a)/2 \rfloor$ in $O(\log n)$ steps using precomputed jumps from LCA. Let x be the vertex at distance t from a.
- Distinguish the case when the vulnerability set contains one or two vertices based on parity of $d(a, b)$. Even – $\{x\}$, odd – $\{x, p(x)\}$ ($p(x)$ is the parent of x).
- The case $d(\ell, a) > d(\ell, b)$ is symmetric.
- Total running time: $O(n \log n + q \log n + n) = O((n + q) \log n)$.

Watchcat Jiskra

Cowpproximation

Cowpproximation

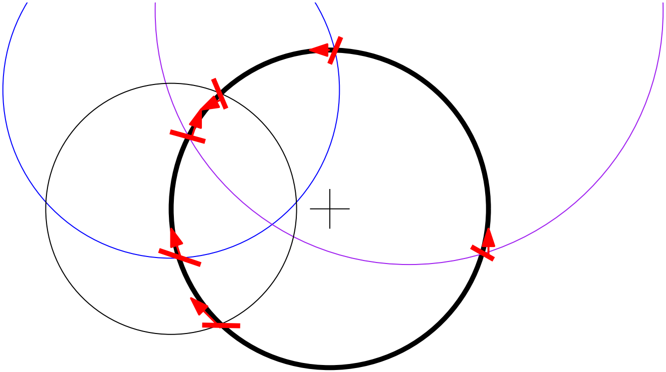
- **Input**: Set of N circles, given by x, y and radius r.
- **Task**: Suppose each circle can travel with speed up to 1 unit of distance per second. In how many seconds can all circles contain a common point?

Cowpproximation

- **Combinatorial solution**:
- **Observation**: We can assume each circle moves at maximum speed and waits at some point if needed.
- Note after t seconds, a circle with radius r can cover exactly the points that are within radius $r + t$.
- We look for minimum t such that if we increase all radii by t, all circles have non-empty intersection.
- Binary search on t.

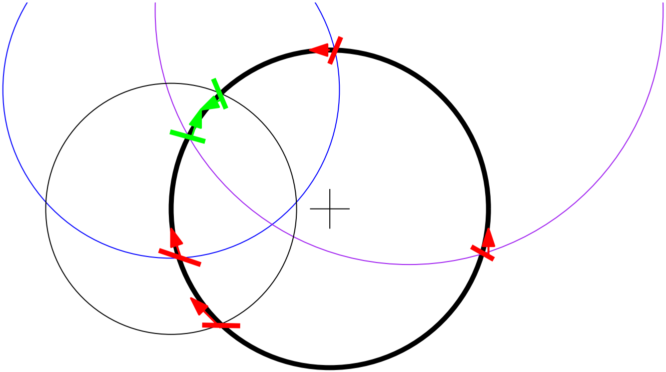
Cowpproximation

- Verify if the set of circles has nonempty intersection:
- Consider a circle C. If the common intersection contains the boundary of C, we can find it as follows.
- The intersections with C give us intervals on the boundary of C.



Cowpproximation

- Verify if the set of circles has nonempty intersection:
- Consider a circle C. If the common intersection contains the boundary of C, we can find it as follows.
- The intersections with C give us intervals on the boundary of C.



Cowpproximation

- **Observation**: Consider function $f(x, y) = max_{C \in \text{circles}} dist((x, y), C)$.
- The minimum of this function is the solution.
- This function is convex.
- Well implemented gradient descend may find the solution quickly.

Reptile eggs

Reptile eggs

- **Input:** Text s and a regular expression r.
- **Output:** Find the size of the longest subsequence of s that is matched with r.

Reptile eggs

- **Input:** Text s and a regular expression r.
- **Output:** Find the size of the longest subsequence of s that is matched with r.

Use dynamic programming!

Reptile eggs

Let A be the automaton accepting the set of strings described by r.

Reptile eggs

Let A be the automaton accepting the set of strings described by r.

- $M[q, i]$ — the longest subsequence on the first i characters of s. (Memory table)

Reptile eggs

Let A be the automaton accepting the set of strings described by r.

- $M[q, i]$ — the longest subsequence on the first i characters of s. (Memory table)
- Let $B(q)$ be a function that is 0 if q is the starting state and $-\infty$ otherwise.

Reptile eggs

Let A be the automaton accepting the set of strings described by r.

- $M[q, i]$ — the longest subsequence on the first i characters of s. (Memory table)
- Let $B(q)$ be a function that is 0 if q is the starting state and $-\infty$ otherwise.
- $M[q, 0] = B(q)$

Reptile eggs

Let A be the automaton accepting the set of strings described by r.

- $M[q, i]$ — the longest subsequence on the first i characters of s. (Memory table)
- Let $B(q)$ be a function that is 0 if q is the starting state and $-\infty$ otherwise.
- $M[q, 0] = B(q)$
- Let Q' be all the possible states q' such that reading character s_i advances A to state q and $1 \leq i \leq |s|$:

$$M[q, i] = max(B(q), M[q, i-1], max_{q' \in Q'}(M[q', i-1]))$$

Reptile eggs

Let A be the automaton accepting the set of strings described by r.

- $M[q, i]$ — the longest subsequence on the first i characters of s. (Memory table)
- Let $B(q)$ be a function that is 0 if q is the starting state and $-\infty$ otherwise.
- $M[q, 0] = B(q)$
- Let Q' be all the possible states q' such that reading character s_i advances A to state q and $1 \leq i \leq |s|$:

$$M[q, i] = max(B(q), M[q, i-1], max_{q' \in Q'}(M[q', i-1]))$$

Output maximum number such that $M[q_f, i]$ is maximized and nonegative, where q_f is a final state. If no such pair of q_f and i does not exist, output -1.

Reptile eggs

Let A be the automaton accepting the set of strings described by r.

- $M[q, i]$ — the longest subsequence on the first i characters of s. (Memory table)
- Let $B(q)$ be a function that is 0 if q is the starting state and $-\infty$ otherwise.
- $M[q, 0] = B(q)$
- Let Q' be all the possible states q' such that reading character s_i advances A to state q and $1 \leq i \leq |s|$:

$$M[q, i] = max(B(q), M[q, i-1], max_{q' \in Q'}(M[q', i-1]))$$

Output maximum number such that $M[q_f, i]$ is maximized and nonegative, where q_f is a final state. If no such pair of q_f and i does not exist, output -1.

Alternatively, the automaton can be represented implicitly by the regular expression. In such case it is enough to consider a position inside the regex instead of a state in the Automaton.

Pigpartite giraffe

Pigpartite giraffe

- **Input**: Small bipartite graph ($n \leq 8$ vertices in each partite).
- **Queries**: Given vertices v, u, add a new x vertex with neighborhood $N(x) = (N(v) \setminus N(u)) \cup (N(u) \setminus N(v))$.
- **Task**: After each query, compute the total sum of distances within the graph.
- Consider the incidence matrix of the graph. The query corresponds to taking a XOR of two rows (equivalently sum $\mod 2$).

Pigpartite giraffe

- Each vertex can be described by the set of its original ancestors (the original 8 vertices in its partite).
- Only 2^8 possible types of vertices in each partite, hence $2 \cdot 2^8$ different type of vertices in total.
- Two vertices of the same type have the same neighborhood.
- **Observation:** Adding a new vertex of an existing type will not change distances between other vertices.

Pigpartite giraffe

Adjacency matrix of a bipartite graph G looks like

$$\mathbf{G} = \begin{array}{c|cc} & A & B \\ \hline A & \mathbf{0} & \mathbf{M} \\ B & \mathbf{M}^\top & \mathbf{0} \end{array}$$

So $\mathbf{M}$ describes our bipartite graph: rows for partite A, columns for partite B.

$$\mathbf{M}_{v,u} = 1 \iff v \in A, u \in B \text{ have an edge.}$$

Pigpartite giraffe

$$\mathbf{M} = \left(\begin{array}{c|cccc} (1) & 1 & 0 & 0 & 0 \\ (2) & 1 & 1 & 1 & 1 \\ (3) & 0 & 0 & 1 & 0 \\ (4) & 0 & 0 & 0 & 1 \end{array}\right)$$

Pigpartite giraffe

$$\mathbf{M} = \left(\begin{array}{c|cccc} (1) & 1 & 0 & 0 & 0 \\ (2) & 1 & 1 & 1 & 1 \\ (3) & 1 & 0 & 1 & 0 \\ (4) & 1 & 0 & 0 & 1 \\ (3 \oplus 4) & 0 & 0 & 1 & 1 \end{array}\right)$$

Pigpartite giraffe

$$\mathbf{M} = \left(\begin{array}{c|cccc} (1) & 1 & 0 & 0 & 0 \\ (2) & 1 & 1 & 1 & 1 \\ (3) & 1 & 0 & 1 & 0 \\ (4) & 1 & 0 & 0 & 1 \\ (3 \oplus 4) & 0 & 0 & 1 & 1 \end{array}\right)$$

Pigpartite giraffe

$$\mathbf{M} = \left(\begin{array}{c|cccc} (1) & 1 & 0 & 0 & 0 \\ (2) & 1 & 1 & 1 & 1 \\ (3) & 1 & 0 & 1 & 0 \\ (4) & 1 & 0 & 0 & 1 \\ (3 \oplus 4) & 0 & 0 & 1 & 1 \\ (3 \oplus 4 \oplus 4 = 3) & 1 & 0 & 1 & 0 \end{array}\right)$$

Pigpartite giraffe

$$\mathbf{M} = \left(\begin{array}{c|cccc} (1) & 1 & 0 & 0 & 0 \\ (2) & 1 & 1 & 1 & 1 \\ (3) & 1 & 0 & 1 & 0 \\ (4) & 1 & 0 & 0 & 1 \\ (3 \oplus 4) & 0 & 0 & 1 & 1 \\ (3) & 1 & 0 & 1 & 0 \end{array}\right)$$

Pigpartite giraffe

$$\mathbf{M} = \left(\begin{array}{c|cccc} (1) & 1 & 0 & 0 & 0 \\ (2) & 1 & 1 & 1 & 1 \\ (3) & 1 & 0 & 1 & 0 \\ (4) & 1 & 0 & 0 & 1 \\ (3 \oplus 4) & 0 & 0 & 1 & 1 \\ (3) & 1 & 0 & 1 & 0 \\ (\bigoplus_{i \in S \subseteq \{1,2,3,4\}} i) & \cdot & \cdot & \cdot & \cdot \end{array}\right)$$

Pigpartite giraffe

$$\mathbf{M} = \left(\begin{array}{c|cccc} & (1) & (2) & (3) & (4) \\ \hline (1) & 1 & 0 & 0 & 0 \\ (2) & 1 & 1 & 1 & 1 \\ (3) & 1 & 0 & 1 & 0 \\ (4) & 1 & 0 & 0 & 1 \\ (3 \oplus 4) & 0 & 0 & 1 & 1 \\ (3) & 1 & 0 & 1 & 0 \end{array}\right)$$

Pigpartite giraffe

$$\mathbf{M} = \left(\begin{array}{c|cc|cc|c}
 & (1) & (2) & (3) & (4) & (1 \oplus 2) \\
\hline
(1) & 1 & 0 & 0 & 0 & 1 \\
(2) & 1 & 1 & 1 & 1 & 0 \\
(3) & 1 & 0 & 1 & 0 & 1 \\
(4) & 1 & 0 & 0 & 1 & 1 \\
(3 \oplus 4) & 0 & 0 & 1 & 1 & 0 \\
(3) & 1 & 0 & 1 & 0 & 1
\end{array}\right)$$

Pigpartite giraffe

- How to efficiently compute the distances?
- First compute pair-wise distances and keep the distance matrix D ($O(n^3)$).
- Keep D small by keeping only one vertex for each type.
- Keep the count for each type.
- We will keep at most $d \leq 2 \cdot 2^n$ vertices in D.
- Also keep track of the total distances for each vertex of D.

Pigpartite giraffe

- When adding a new vertex v:
- If v has a **new type**:
  - Compute the distance from v to all others with BFS ($O(d^2)$).
  - Update D for all other vertices: for each pair $x, y \in V$, check if $dist(x, v) + dist(v, y) < dist(x, y)$ ($O(d^2)$).
- If v has an **existing type**:
  - Just update the count of v's type.
  - Sum the distances from each vertex ($O(d)$).
    - Be careful about vertices of the same type as v! Especially if v is just the second vertex of that type.
- A new type appears at most $2 \cdot 2^n$ times.
- Let's evaluate the total runtime.
- $O(n^3 + 2^n d^2 + Qd)$ with $d = O(2^n)$.
- $O(n^3 + 2^{3n} + Q2^n)$ with $n \leq 8, Q \leq 10^5$.
- $\approx 2^9 + 2^{24} + 2^{25}$