CTU Open 2019

Presentation of solutions

October 19, 2019

Beer Bill

- Task: Given a pub bill compute the bill total rounded up to nearest 10s.

```
||||                                  540,-
123,-|||
```

Beer Barrels

- Task: Given all numbers made of K digits with values A or B, sum up all occurences of digit C.

Beer Barrels

- Task: Given all numbers made of K digits with values A or B, sum up all occurences of digit C.

If $A = 4, B = 5$ and $C = 4$, the number of C's in the following set of numbers is 12.

```
444 455
445 545
454 554
544 555
```

Beer Barrels

- Task: Given all numbers made of K digits with values A or B, sum up all occurences of digit C.

If $A = 4, B = 5$ and $C = 4$, the number of C's in the following set of numbers is 12.

```
444  455
445  545
454  554
544  555
```

- Solve special cases separately.
  - If $C \neq A$ and $C \neq B$, answer is 0,
  - if $A = B = C$ answer is 1.

Beer Barrels

- Task: Given all numbers made of K digits with values A or B, sum up all occurences of digit C.

If $A = 4, B = 5$ and $C = 4$, the number of C's in the following set of numbers is 12.

```
444  455
445  545
454  554
544  555
```

- Solve special cases separately.
  - If $C \neq A$ and $C \neq B$, answer is 0,
  - if $A = B = C$ answer is 1.
- Note that every number can be mirrored by exchanging A and B

- Note that every number can be mirrored by exchanging A and B

```
444 <-> 555
445 <-> 554
454 <-> 545
544 <-> 455
```

- Note that every number can be mirrored by exchanging A and B

```
444 <-> 555
445 <-> 554
454 <-> 545
544 <-> 455
```

- So for each pair there are exactly K occurrences of the digit.

- Note that every number can be mirrored by exchanging A and B

```
444 <-> 555
445 <-> 554
454 <-> 545
544 <-> 455
```

- So for each pair there are exactly K occurrences of the digit.
- Answer is $K \cdot 2^K/2$.

Complexity $O(K)$

Beer Vision

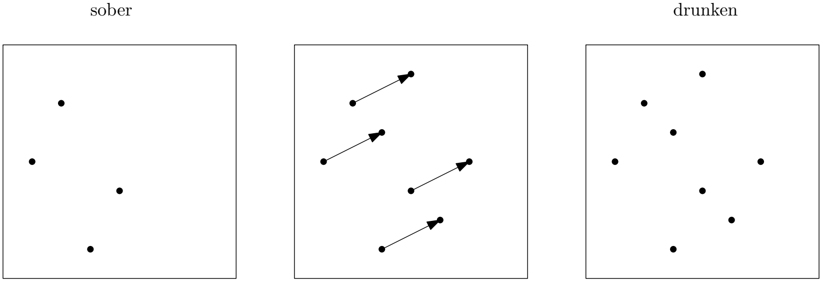
- A drunken image (points in 2D) is created from a sober image by shifting the sober image and merging the original and shifted sober image.

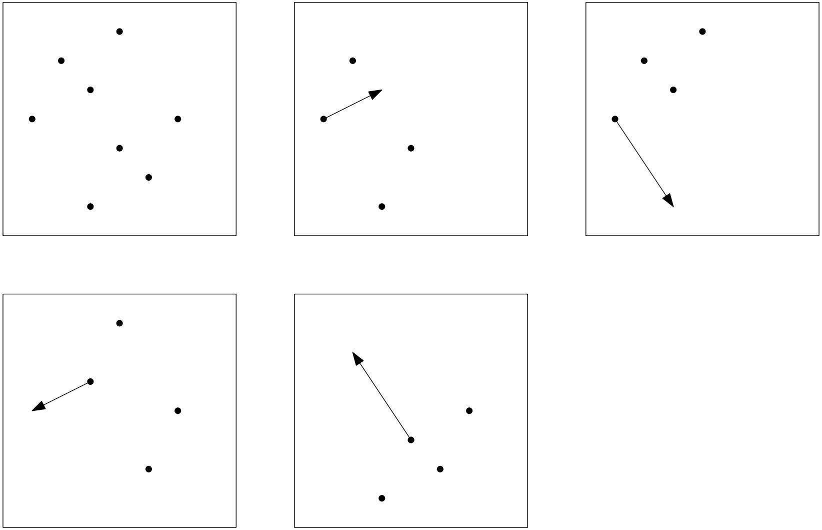
Beer Vision

- A drunken image (points in 2D) is created from a sober image by shifting the sober image and merging the original and shifted sober image.
- Task: Given a drunken image, count the number of different vectors which can be used to create the drunken image from a sober image.

Beer Vision

- A drunken image (points in 2D) is created from a sober image by shifting the sober image and merging the original and shifted sober image.
- Task: Given a drunken image, count the number of different vectors which can be used to create the drunken image from a sober image.



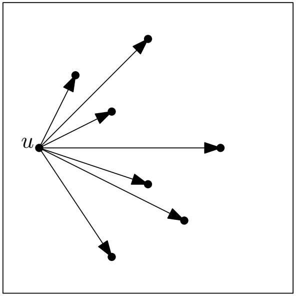


- Each vertex must be in either the original sober image or in its copy.

- Each vertex must be in either the original sober image or in its copy.
- Pick vertex v; for any fixed vector if it is in the original image, then it must have a copy present in the drunken image. If it is an image, it must have its original in the drunken image.

- Each vertex must be in either the original sober image or in its copy.
- Pick vertex v; for any fixed vector if it is in the original image, then it must have a copy present in the drunken image. If it is an image, it must have its original in the drunken image.
- Solution:
    - Try for each vector starting at a fixed vertex ending in all other vertices.

- Each vertex must be in either the original sober image or in its copy.
- Pick vertex v; for any fixed vector if it is in the original image, then it must have a copy present in the drunken image. If it is an image, it must have its original in the drunken image.
- Solution:
  - Try for each vector starting at a fixed vertex ending in all other vertices.
  - The vector v is good if for each vertex u there is either $u + v$ or $u - v$ in the drunken image.
  - complexity $O(n^2 \cdot \text{hash})$



Beer Mugs

- Let us have look on how such permutation shall look like:

Beer Mugs

- Let us have look on how such permutation shall look like:
- Obviously it must be a palindrom:
- This means that every that all - but (at most) one - characters must be included even number of times.

Beer Mugs

- Let us have look on how such permutation shall look like:
- Obviously it must be a palindrom:
- This means that every that all - but (at most) one - characters must be included even number of times.
- Lets find a bit-mask for each index: I-th bit is on, if I-th character is odd number of times in string [0:I]

Beer Mugs

- Let us have look on how such permutation shall look like:
- Obviously it must be a palindrom:
- This means that every that all - but (at most) one - characters must be included even number of times.
- Lets find a bit-mask for each index: I-th bit is on, if I-th character is odd number of times in string [0:I]
- Now observe, that if we XOR two such masks and the result contains zero or one 1bit, it is a valid substring (between those indices [exclude/include])

Beer Mugs

- Let us have look on how such permutation shall look like:
- Obviously it must be a palindrom:
- This means that every that all - but (at most) one - characters must be included even number of times.
- Lets find a bit-mask for each index: I-th bit is on, if I-th character is odd number of times in string [0:I]
- Now observe, that if we XOR two such masks and the result contains zero or one 1bit, it is a valid substring (between those indices [exclude/include])
- Now we can simply go from left to right, putting such masks to a map (or better an array) while checking whether there exists a previous occurence which is either same or with one bit off.

Beer Mugs

- Let us have look on how such permutation shall look like:
- Obviously it must be a palindrom:
- This means that every that all - but (at most) one - characters must be included even number of times.
- Lets find a bit-mask for each index: I-th bit is on, if I-th character is odd number of times in string [0:I]
- Now observe, that if we XOR two such masks and the result contains zero or one 1bit, it is a valid substring (between those indices [exclude/include])
- Now we can simply go from left to right, putting such masks to a map (or better an array) while checking whether there exists a previous occurence which is either same or with one bit off.

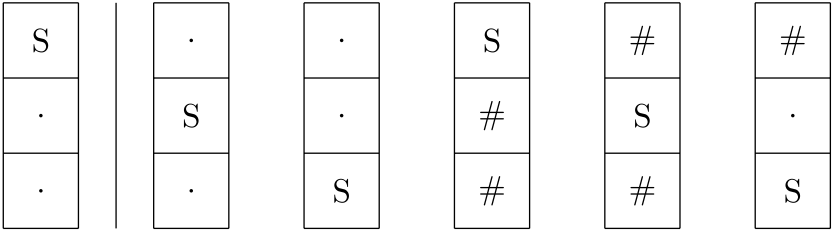
Complexity $O(N \cdot |\alpha\beta| + 2^{|\alpha\beta|})$

Screamers in the Storm

- Simple implementation task
- Just implement and simulate described process
- Pay close attention to when particular events occur (death of starvation, grass grows after 3 turns after the beginning of the game . . .)

Screamers in the Storm

- Simple implementation task
- Just implement and simulate described process
- Pay close attention to when particular events occur (death of starvation, grass grows after 3 turns after the beginning of the game . . .)



Beer Can Game

- Observation: Insert can / Remove can are standard operations used in edit distance calculation.
- The tricky part is the last operation, expanding the token.
- Observation: We have to expand all tokens to beer cans anyway so we can separate adding number of expansions to the result and expansions themselves.

Beer Can Game

- Expanded token can be used as any letter so we'll expand it to some wildcard symbol and postpone the decision of specific symbols.

Beer Can Game

- Expanded token can be used as any letter so we'll expand it to some wildcard symbol and postpone the decision of specific symbols.
- Example:
  - **edit3tance** becomes **edit???tance**
  - Plus we need to add 1 to the result as we did one expansion.
- We apply the same approach to both input lines.

Beer Can Game

- Expanded token can be used as any letter so we'll expand it to some wildcard symbol and postpone the decision of specific symbols.
- Example:
  - **edit3tance** becomes **edit???tance**
  - Plus we need to add 1 to the result as we did one expansion.
- We apply the same approach to both input lines.
- The rest is to adjust edit distance calculation to accept wildcards. Allow match for a combination wildcard and a letter and also for two wildcars at the leading positions of the lines.

Beer Can Game

- First input line can be up to 10 000 characters long and include at most 100 digits. After applying expansion it can be up to $N = 10\,800$ characters long.
- Second input line can be up to 1 000 characters long and include at most 100 digits. After applying expansion it can be up to $M = 1\,800$ characters long.

Beer Can Game

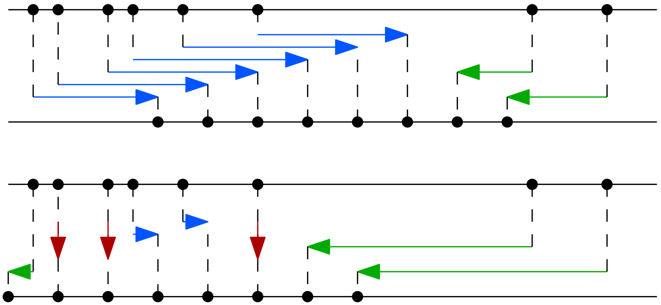
- First input line can be up to 10 000 characters long and include at most 100 digits. After applying expansion it can be up to $N = 10\,800$ characters long.
- Second input line can be up to 1 000 characters long and include at most 100 digits. After applying expansion it can be up to $M = 1\,800$ characters long.
- We'll calculate edit distance using dynamic programming in $O(NM)$.
- Overall time complexity is $\mathcal{O}(NM)$.

Beer Marathon

- Key observation: Imagine we have an already formed sequence of booths with correct distances between each other
- If we shift this whole solution by a single unit of distance to the left, we save the distance equal to the number of booths that were moved to this solution from the left, minus the distance equal to the number of booths moved there from the right
- ...and vice versa
- It is thus profitable to repeatedly shift the solution to the side from which the larger number of booths was obtained
- Therefore, the optimal solution is reached when the number of booths from either side is equal
- $\Rightarrow$ the function of the result w.r.t. the position of the sequence is unimodal
- We can find the optimal solution with e.g. ternary search
- Time complexity (also including sorting): $\mathcal{O}(N \log N)$

Beer Marathon

- Another solution:



Sixpack

- The sum of all numbers across all the sixpacks must remain constant.
- When moving towards the next sixpack we dispose of one whole column and add another one.
- Observation: The sum of numbers in the exchanged columns must be the same.

Sixpack

- One of the ways to solve this problem is to set 3 fixed sums for the columns, as the sum of column is bound to repeat during an exchange of columns.
- We can precompute number of ways to reach the sum of X in the added column, which obviously depends on how many numbers are pre-filled in this column.
- We will loop through each column for every combination of fixed sums and calculate number of ways to fill out the column.
- And through that we will also check for validity as otherwise the number of ways to reach certain value will be equal to zero.

Sixpack

- Maximal possible sum for first (modulo 3) column is 18. For the second column the possible sum goes also up to 18. And for the third one we will compute it from the first two.
- We are doing an iteration through all columns.
- Time complexity of this solution is $\mathcal{O}(N)$

Beer Coasters

Beer Coasters

- Task: Compute area of a beer coaster (rectangle) which is covered by a beer (circle).

Beer Coasters

- Task: Compute area of a beer coaster (rectangle) which is covered by a beer (circle).
- Compute area of rectangle-circle intersection.

Beer Coasters

- Task: Compute area of a beer coaster (rectangle) which is covered by a beer (circle).
- Compute area of rectangle-circle intersection.
- Approaches:
  - Either try to differentiate every possible case,

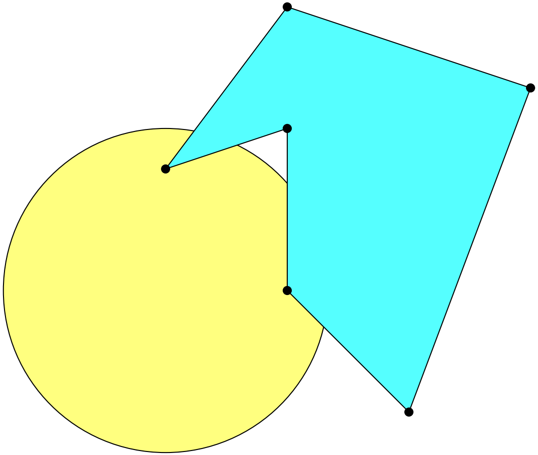
Beer Coasters

- Task: Compute area of a beer coaster (rectangle) which is covered by a beer (circle).
- Compute area of rectangle-circle intersection.
- Approaches:
  - Either try to differentiate every possible case,
  - or write generalized solution to the problem.

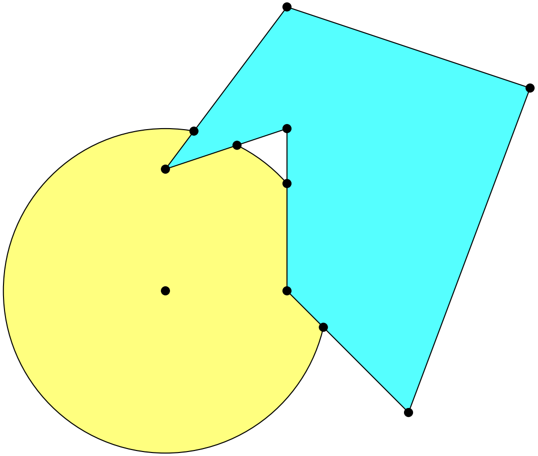
Beer Coasters

- Task: Compute area of a beer coaster (rectangle) which is covered by a beer (circle).
- Compute area of rectangle-circle intersection.
- Approaches:
  - Either try to differentiate every possible case,
  - or write generalized solution to the problem.
- Alter the standard polygon area algorithm.

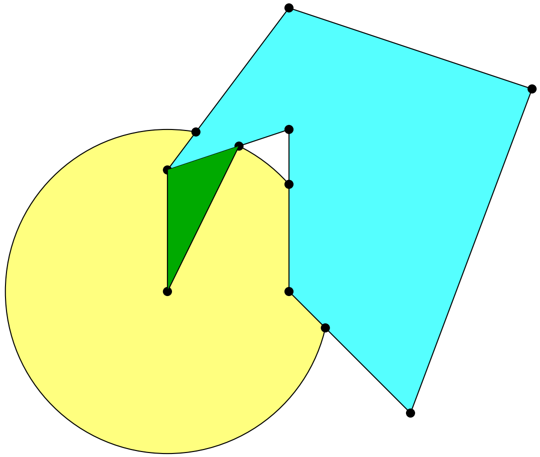
Beer Coasters



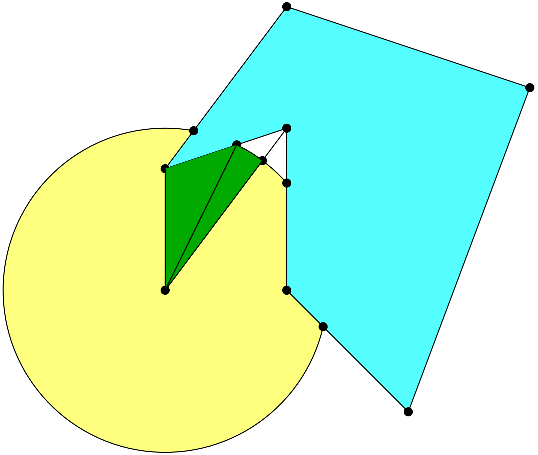
Beer Coasters



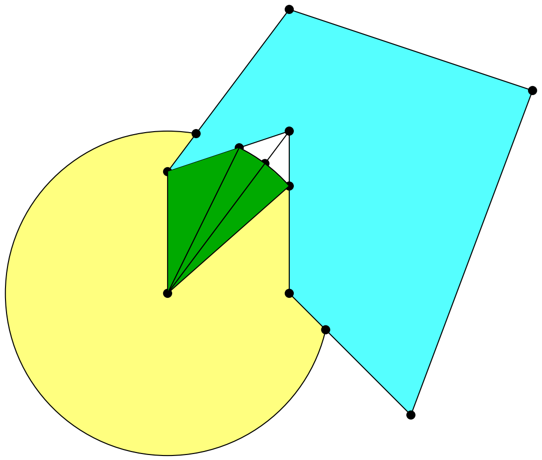
Beer Coasters



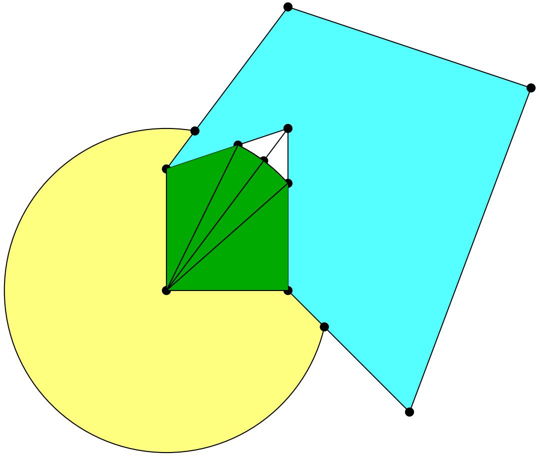
Beer Coasters



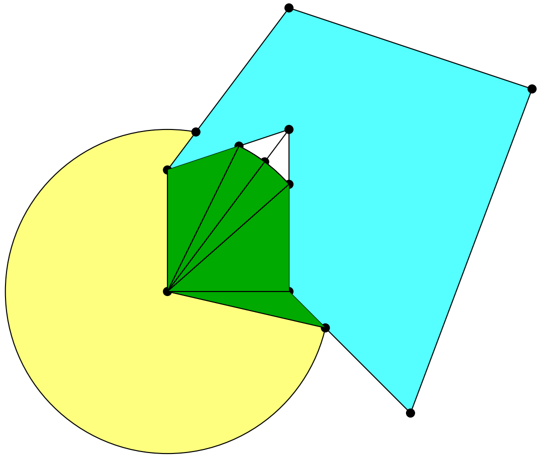
Beer Coasters



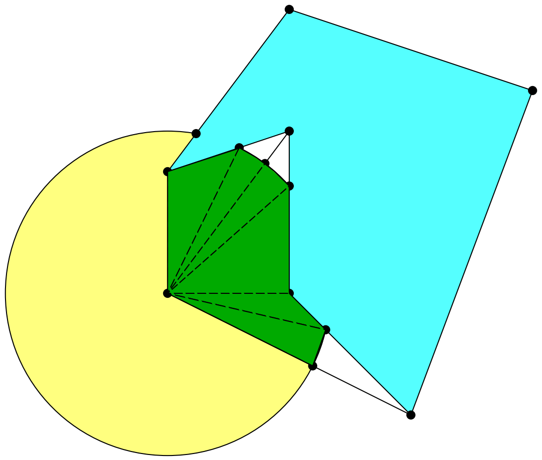
Beer Coasters



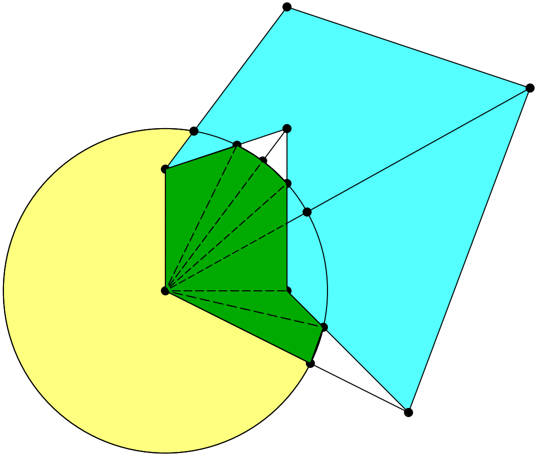
Beer Coasters



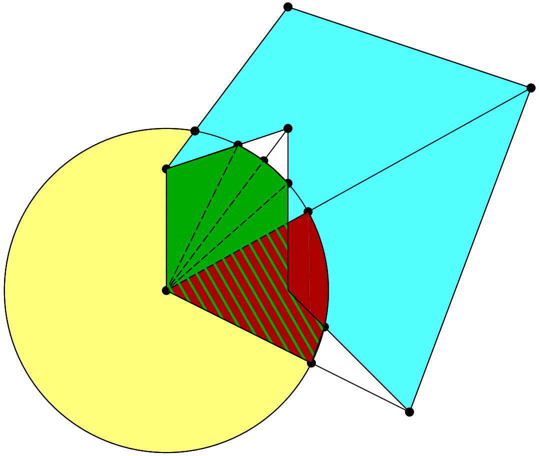
Beer Coasters



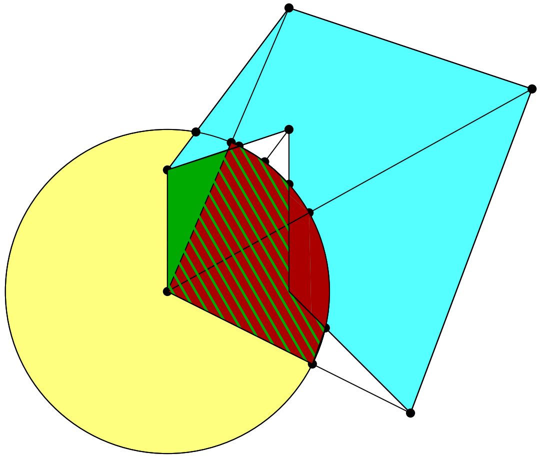
Beer Coasters



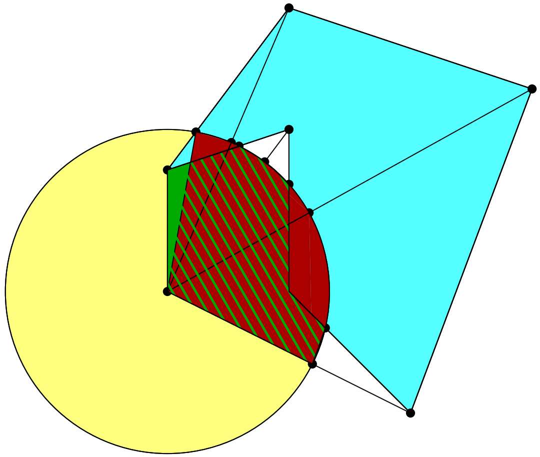
Beer Coasters



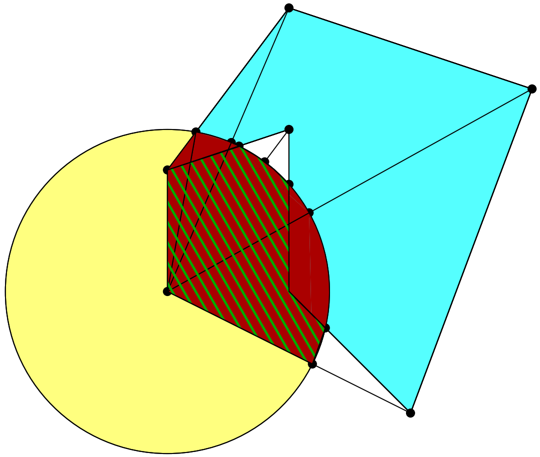
Beer Coasters



Beer Coasters



Beer Coasters



Beer Flood System

- We have a DAG with a single source and a single sink
- Therefore every vertex is reachable from the source and the sink is reachable from every vertex
- The goal is to remove as many edges as possible, while still preserving the above conditions
- How does the remaining graph look like?

Beer Flood System

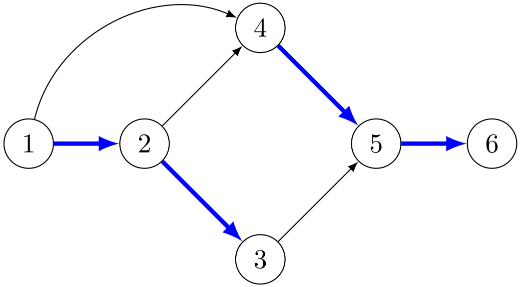
- We have a DAG with a single source and a single sink
- Therefore every vertex is reachable from the source and the sink is reachable from every vertex
- The goal is to remove as many edges as possible, while still preserving the above conditions
- How does the remaining graph look like?
- There must be at least a single incoming and a single outcoming edge for every vertex (except for the source and the sink)
- We would like to do this with as little edges as possible
- In the best case, we want to fulfill the incoming edge requirement of some vertex and the outcoming edge requirement of another vertex with a single edge

Beer Flood System

- Therefore we want to find as many such pairs of vertices as possible
- $\Rightarrow$ we perform maximum bipartite matching

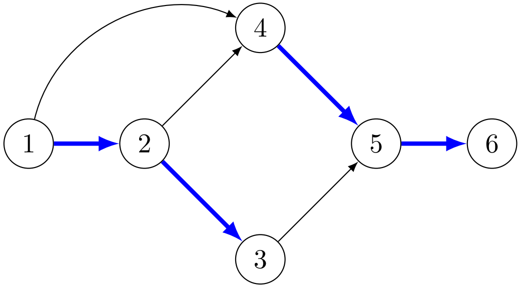
Beer Flood System

- Therefore we want to find as many such pairs of vertices as possible
- $\Rightarrow$ we perform maximum bipartite matching



Beer Flood System

- Therefore we want to find as many such pairs of vertices as possible
- $\Rightarrow$ we perform maximum bipartite matching



- There still may be vertices with either incoming or outcoming edge requirement unsatisfied
- We fulfill each of these with an arbitrary edge
- Sufficient time complexity: $\mathcal{O}(NM)$

Thank you for your attention!