Open the New Frontiers of Science and Technology

Japan Advanced Institute of Science and Technology

Characteristics of Education at JAIST

First Independent National Graduate Institute without Undergraduate Division

- Founded in October 1990 as the first independent national graduate school, to carry out graduate education based on research at the highest level in advanced science and technology.

Use of English as a Primary Language for Instruction and Research

- Being the most internationalized national university in Japan with 32% of the students and 20% of the faculty from overseas, JAIST uses English as a primary language for instruction and research

Admission Criteria for People of Diverse Backgrounds

- Valuing the motivation of the applicants most as demonstrated in the personal interview.

Systematic Graduate Education

- Carefully and systematically designed coursework-oriented curriculum.

Development of Human Resources for Society

- Training students in a specialized field combined with interdisciplinary knowledge of related disciplines.

Outstanding Faculty

- Global-standard excellent faculty and researchers with diverse backgrounds.

Collaboration with Society and Industry

- Working with the regional community, as well as industries worldwide, by promoting collaborative research and accepting commissioned research.

Two important key words

- **Intellectual Toughness**
  - “Intellectual toughness is an ability to think logically based on solid understanding of essential knowledge and skills in more than one academic field. It also requires an ability of coping with any unknown situation and that of communicating with others regardless of difference in cultural backgrounds. Intellectual toughness necessitates a strong will power not to give up in difficult situations, too.”
- **Importance of Change**
  - “To become a matured scientist or engineer, it is important to change something intentionally, which is also related to intellectual toughness. ………… Whatever major students might have pursued, it is very important for them to expand their research interest beyond their specialized field.”

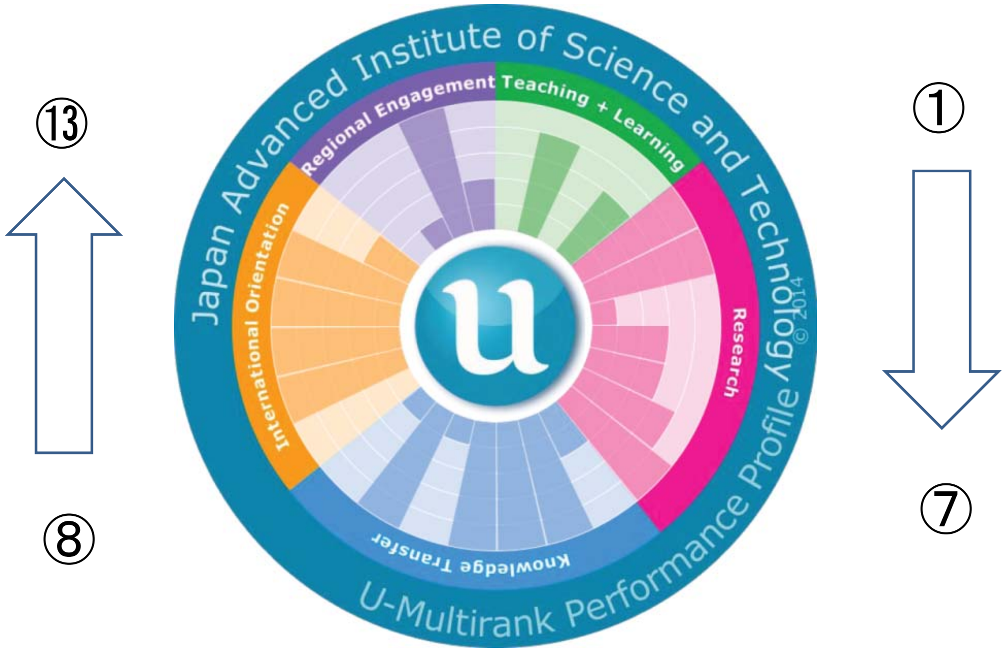
Excerpts from the President Message in JAIST Home Page

JAIST in World Ranking

May 23, 2015

In U-Multirank released by EU in 2015, JAIST acquired excellent ratings in the field of Research and the field of International Orientation, thanks to high achievement in research publications, external research income, foreign language programs, international doctoral degrees, and so on. JAIST was ranked next to the University of Tokyo in the field of research among the participating higher educational institutions in Japan.

The evaluation result "Sunburst Chart" that was provided by U-Multirank.



Research	
External research income	①
Publication output (size-normalised)	②
Art related output	③
Citation rate	④
Top cited papers	⑤
Interdisciplinary publications	⑥
Post-doc positions	⑦

International Orientation	
Foreign language BA programs	⑧
Foreign language MA programs	⑨
Student mobility	⑩
International academic staff	⑪
International doctorate degrees	⑫
International joint publications	⑬

Comment:Research (pink) and International Orientation (orange) were investigated based on indicators shown on the right, and each got a high evaluation result.

Divergent Environment

- Roughly speaking, usual Japanese students, foreign students, and students with jobs are equally many.
- About 20% of professors come from foreign countries.
- English is used as a primary language for instruction and research.

Three important areas of JAIST

Knowledge Science

Information Science

Materials Science

Three areas are unified

- Promote fusion research
- Free innovative ideas independent of traditional studies
- Demands for broader experiences and knowledge

Nine Academic Fields

1. Human Life Design, 2. Knowledge Management,
3. Security・Network, 4. Intelligent Robotics,
5. Game Entertainment, 6. Environment and Energy
7. Materials and Chemistry, 8. Applied Physics,
9. Bioscience and Biotechnology

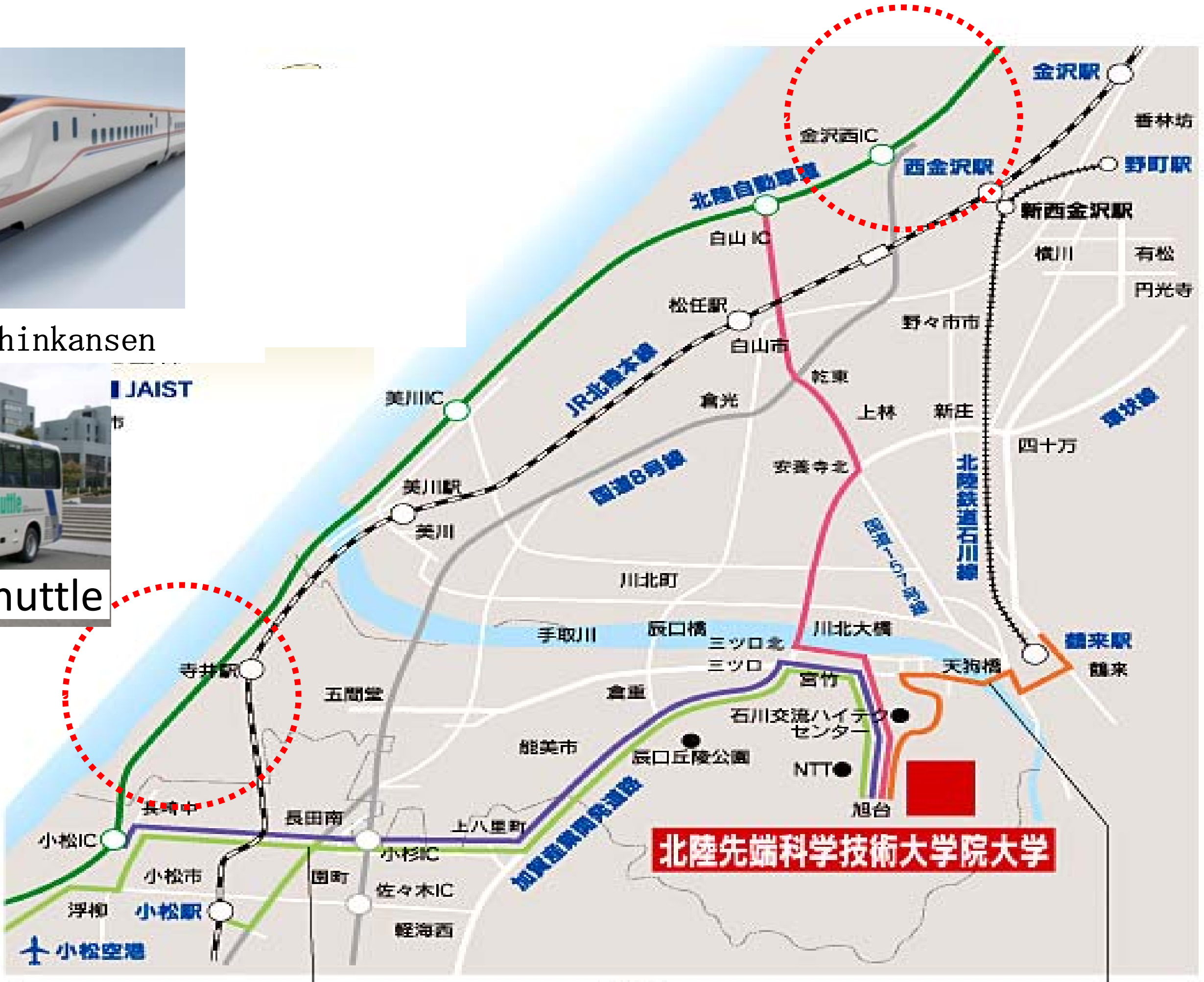
Access

Hokuriku Shinkansen

JAIST-Shuttle

JAIST

CarSharing



小松空港およびJR小松駅から本学までの間には
送迎車「JAIST Shuttle」(小松駅線、小松空港線)
(予約制)を運行しています。

北陸鉄道鶴来駅から本学までの間には
連絡バス「JAIST Shuttle」(鶴来線)(無料)
を運行しています。

Finding Fun of Algorithm Research
---Algorithm in Everyday Life---

JAIST SAST 2015

Japan Advanced Institute of Science and Technology
President
Tetsuo Asano

What is an algorithm?

- Algorithm originates from an Arabic researcher Al-Khwarizmi who edited a textbook on Algebra in 9th Century.
- It describes how to solve a problem using a computer.
- It must be efficient.
- It is similar to cooking recipe, but it cannot contain any ambiguous expression (e.g., just a small amount of salt).

Computing sales amount

Given a set of sales amount for a month, compute the amount of sales for a given period of days.

Daily sales

How much sales from the 6th to 15th?

1	2	3	4	5	6	7	8	9	10
12712	15132	19867	17653	18890	21230	22329	21983	31823	32331

11	12	13	14	15	16	17	18	19	20
30124	32109	31207	29003	28090	29708	32334	32109	32980	33782

21	22	23	24	25	26	27	28	29	30
29201	32038	33201	27903	26700	26903	29806	32105	29503	32317

Given a set of sales amount for a month, compute the amount of sales for a given period of days.

Daily sales

How much sales from the 6th to 15th?

1	2	3	4	5	6	7	8	9	10
12712	15132	19867	17653	18890	21230	22329	21983	31823	32331
11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
30124	32109	31207	29003	28090	29708	32334	32109	32980	33782
21	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
29201	32038	33201	27903	26700	26903	29806	32105	29503	32317

Algorithm 1 :

Compute the sum of sales from A-day to B-day.

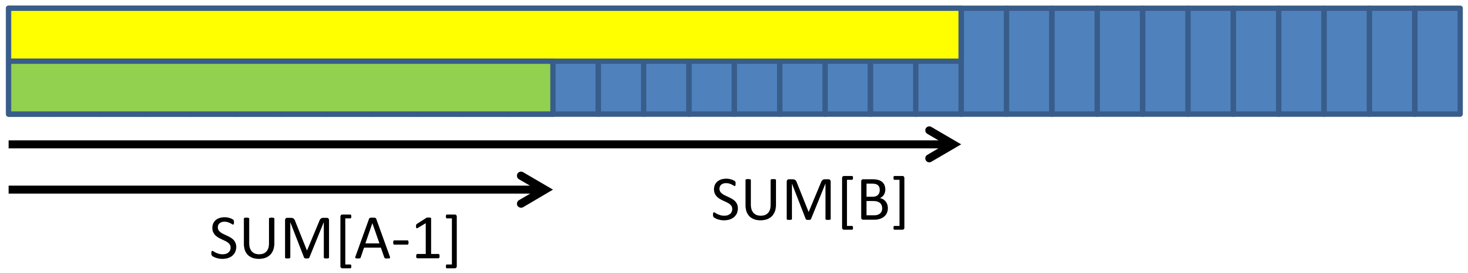
Efficiency of the algorithm :

B-A Additions are needed. Mental arithmetic is hard for 10 days.

Given a set of sales amount for a month, compute the amount of sales for a given period of days.

Algorithm 2:

Compute the sums of sales from the first day of the month.
SUM[A] = the sum of sales from the 1st day to the A-th day.



Once we compute them, we obtain the sum between A and B as SUM[B] - SUM[A-1].

Efficiency of the algorithm:

We need 30 additions in advance, but we can calculate the sum between A and B days just in a single subtraction.

sales amount

the sum between 6th and 15th days?

1	2	3	4	5	6	7	8	9	10
12712	15132	19867	17653	18890	21230	22329	21983	31823	32331

11	12	13	14	15	16	17	18	19	20
30124	32109	31207	29003	28090	29708	32334	32109	32980	33782

21	22	23	24	25	26	27	28	29	30
29201	32038	33201	27903	26700	26903	29806	32105	29503	32317

Computed sums

SUM[15] - SUM[6-1] = 364483 - 84254=280229

1	2	3	4	5	6	7	8	9	10
12712	27833	47711	65364	84254	105484	127813	149796	181619	213950

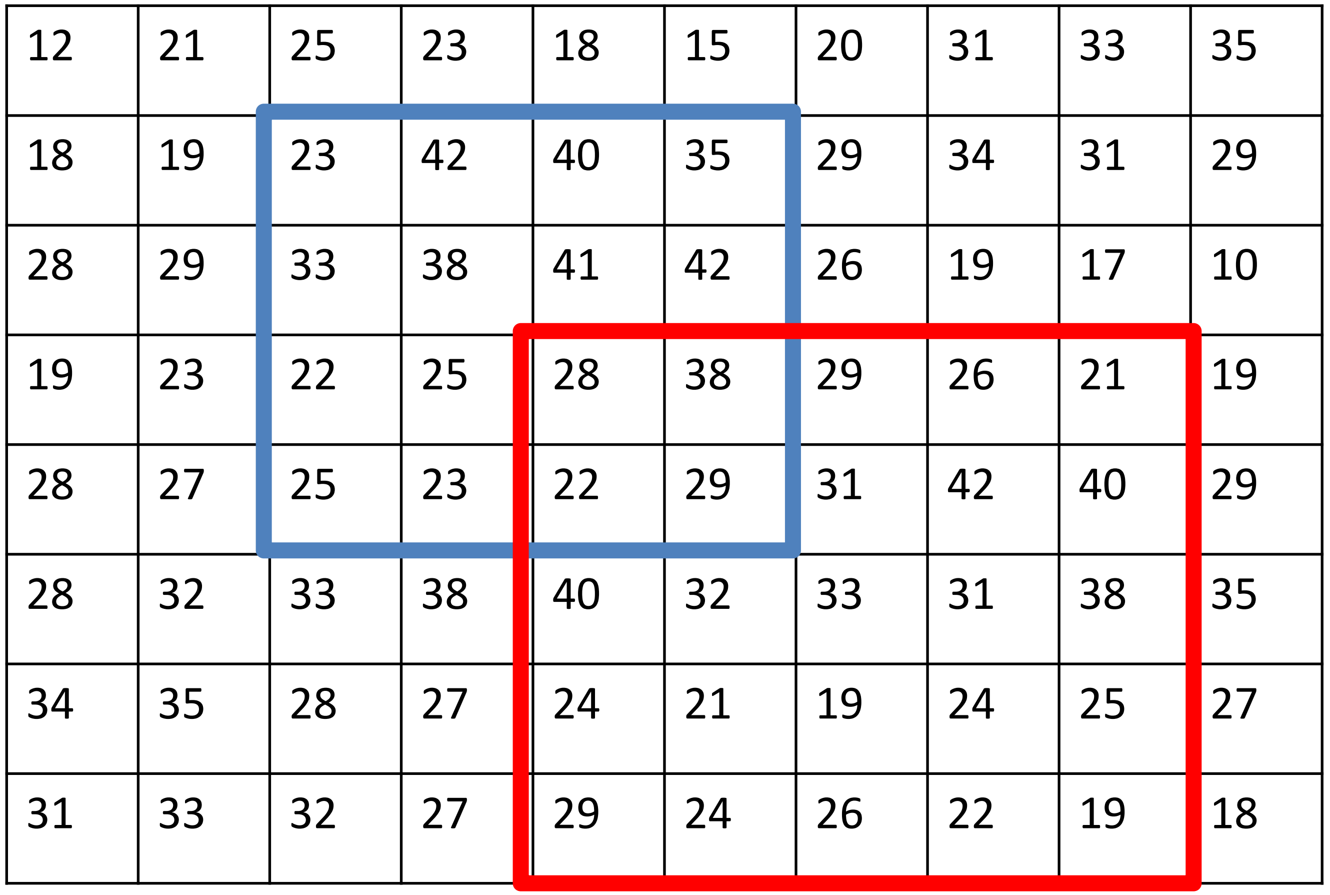
11	12	13	14	15	16	17	18	19	20
244074	276183	307390	336393	364483	394191	426525	458634	491614	525396

21	22	23	24	25	26	27	28	29	30
554597	586635	619836	647739	674439	701342	731148	763253	792756	825073

This algorithm is effective in a situation where sums of sales are asked for many different days.

For example,
Assume that we have a set of million data.
We want to compute the sum of data for any given two days.
Simple algorithm: needs as many additions as the width of intervals.
This algorithm: Independently of how wide an interval is. Just two subtractions are needed for any interval.

Extension of the problem:
Given a 2-dimensional table, compute the sum of values in an arbitrarily given rectangular region.



12	21	25	23	18	15	20	31	33	35
18	19	23	42	40	35	29	34	31	29
28	29	33	38	41	42	26	19	17	10
19	23	22	25	28	38	29	26	21	19
28	27	25	23	22	29	31	42	40	29
28	32	33	38	40	32	33	31	38	35
34	35	28	27	24	21	19	24	25	27
31	33	32	27	29	24	26	22	19	18

Extension of the problem:
Given a 2-dimensional table, compute the sum of values in an arbitrarily given rectangular region.

						20	31	33	35
						29	34	31	29
						26	19	17	10
						29	26	21	19
						31	42	40	29
28	32	33	38	40	32	33	31	38	35
34	35	28	27	24	21	19	24	25	27
31	33	32	27	29	24	26	22	19	18

Extension of the problem:
Given a 2-dimensional table, compute the sum of values in an arbitrarily given rectangular region.

🟩	🟩	🟩	🟩	🟩	🟩	20	31	33	35
18	19	23	42	40	35	29	34	31	29
28	29	33	38	41	42	26	19	17	10
19	23	22	25	28	38	29	26	21	19
28	27	25	23	22	29	31	42	40	29
28	32	33	38	40	32	33	31	38	35
34	35	28	27	24	21	19	24	25	27
31	33	32	27	29	24	26	22	19	18

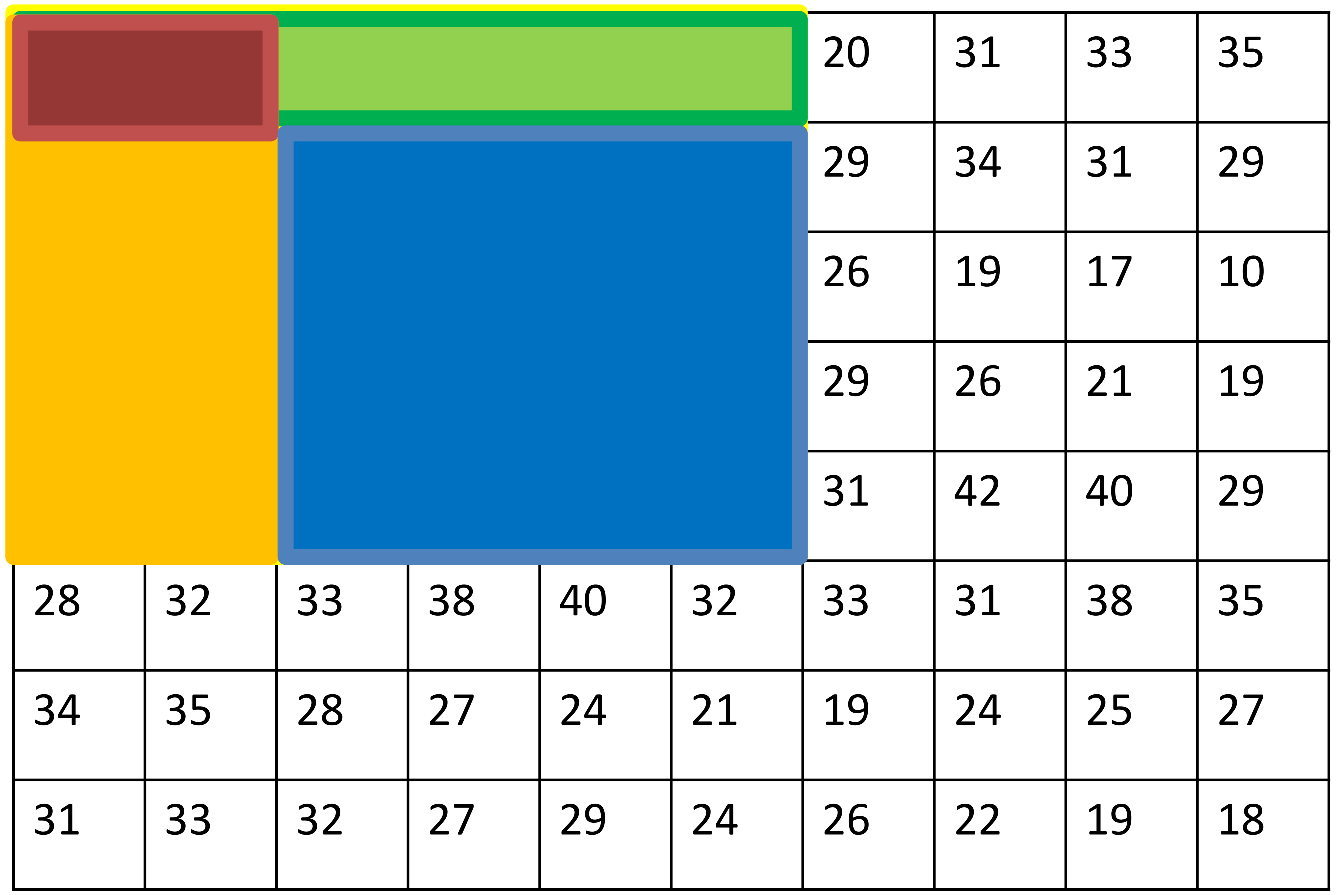
Extension of the problem:
Given a 2-dimensional table, compute the sum of values in an arbitrarily given rectangular region.

		25	23	18	15	20	31	33	35
		23	42	40	35	29	34	31	29
		33	38	41	42	26	19	17	10
		22	25	28	38	29	26	21	19
		25	23	22	29	31	42	40	29
28	32	33	38	40	32	33	31	38	35
34	35	28	27	24	21	19	24	25	27
31	33	32	27	29	24	26	22	19	18

Extension of the problem:
Given a 2-dimensional table, compute the sum of values in an arbitrarily given rectangular region.

[redacted]	[redacted]	25	23	18	15	20	31	33	35
18	19	23	42	40	35	29	34	31	29
28	29	33	38	41	42	26	19	17	10
19	23	22	25	28	38	29	26	21	19
28	27	25	23	22	29	31	42	40	29
28	32	33	38	40	32	33	31	38	35
34	35	28	27	24	21	19	24	25	27
31	33	32	27	29	24	26	22	19	18

Extension of the problem:
Given a 2-dimensional table, compute the sum of values in an arbitrarily given rectangular region.



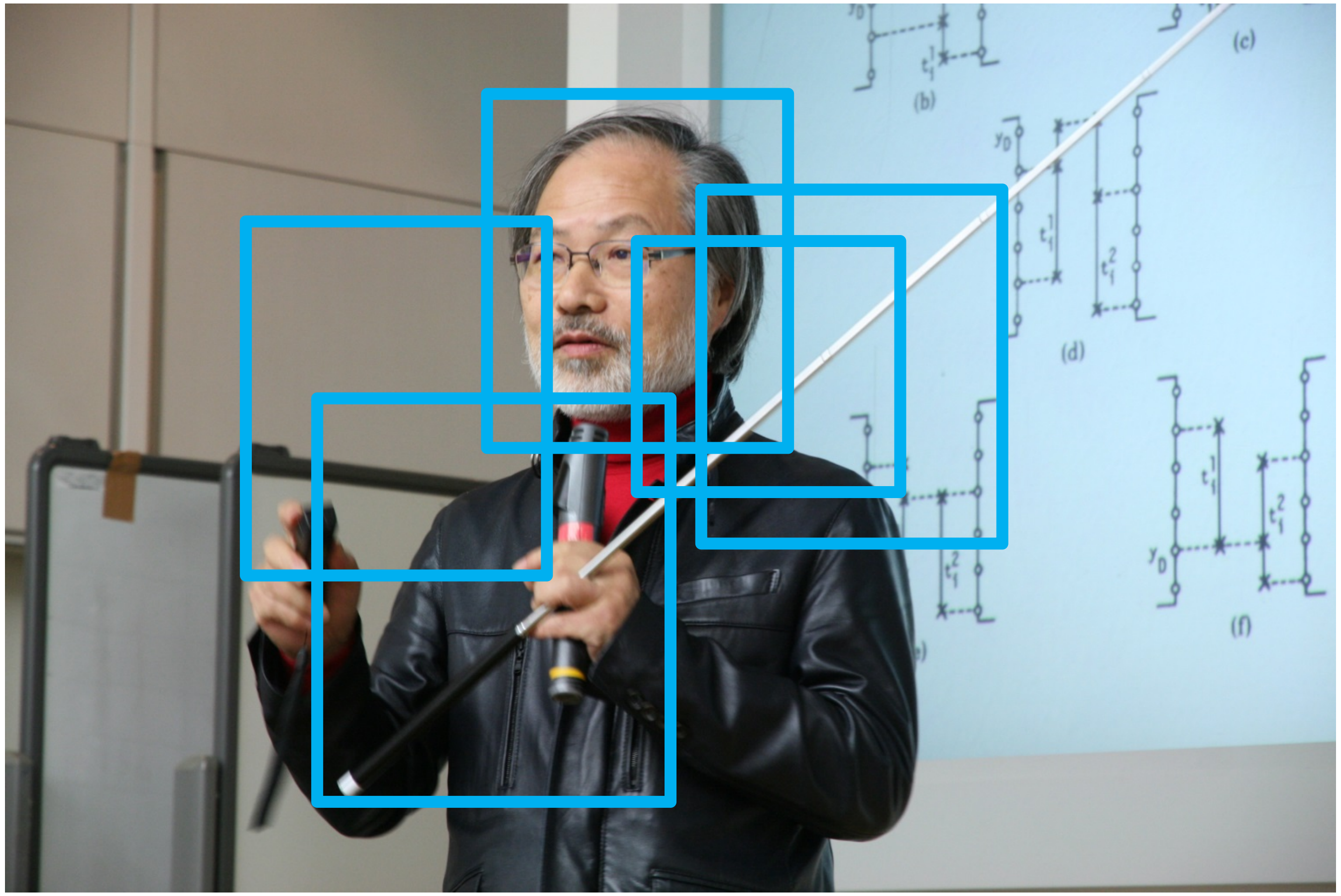
						20	31	33	35
						29	34	31	29
						26	19	17	10
						29	26	21	19
						31	42	40	29
28	32	33	38	40	32	33	31	38	35
34	35	28	27	24	21	19	24	25	27
31	33	32	27	29	24	26	22	19	18

For any given rectangle, only three operations suffice.

Application:

Recognizing a human face in digital camera:

known as the Viola-Jones method



Efficiency of the algorithm

For an image of 3,300x3,300 pixels,

the number of candidates for a valid square is given by

the number of candidates of the upper left corners

= 10 million candidates

Sizes of a square = 300 different sizes from 200 to 500

∴We have to consider 3 billion squares.

We have to choose a right one after examining 3 billion squares which matches a given human face.

It is impossible to take time for each square.

If we simply add elements

we need 40,000 calculations for size of 200 x 200

or 250,000 calculations for size of 500 x 500.

If we compute additions in advance,

only three operations are needed independently of square size.

Why does a printer produce a beautiful color picture from data in digital camera?

What is *Digital Halftoning?*

Continuous-tone image

with 256 (8 bits) brightness levels

↓

Binary image consisting of black and white dots

Color Image

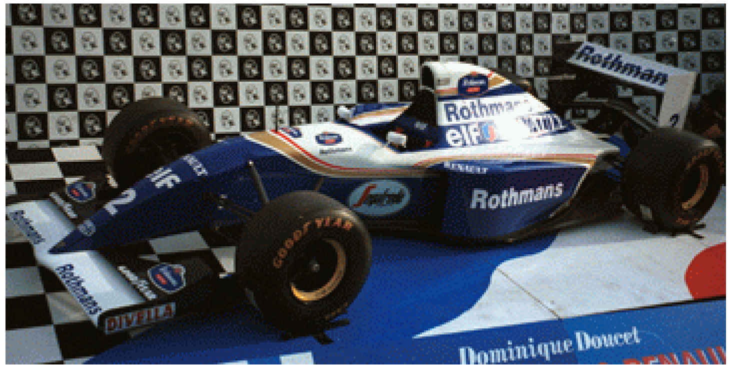
representing each level in **RGB** by 8 bits

↓

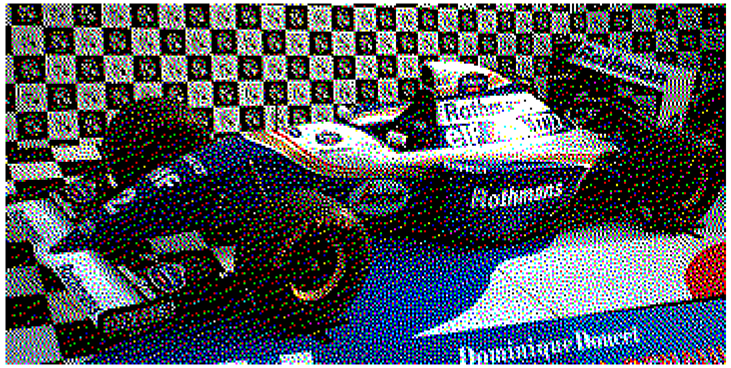
one bit for each of R, G, and B

(8 colors in total)

Digital Halftoning: An example



Input image of 16,700,000 colors (8 bits for R, G, B)



Halftoned image of 8 colors (1 bit for R, G, B)

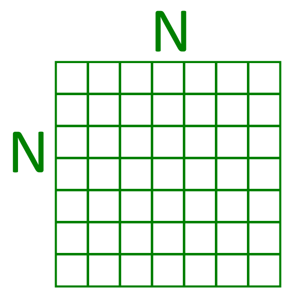
Problem Formulation

Input: A = (a(i, j)), real-valued matrix

$0 \le a(i, j) \le 1$

Output: B = (b(i, j)), binary matrix

b(i,j) = 0 or 1



NxN matrix

Assumption:

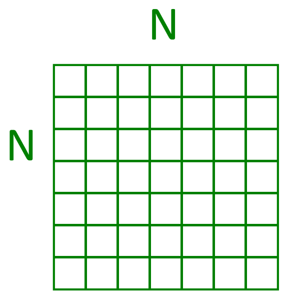
A and B are of the same size.

Criterion:

A binary image B should look similar to an input continuous-tone image A.

Problem Formulation

Input : A matrix of real values between 0 and 1
A = (a(i,j)), a real value between 0 and 1



Output: a matrix consisting of 0 and 1
B = (b(i,j)), 0 or 1

To deal with a color image, we binarize each of R,G, B.

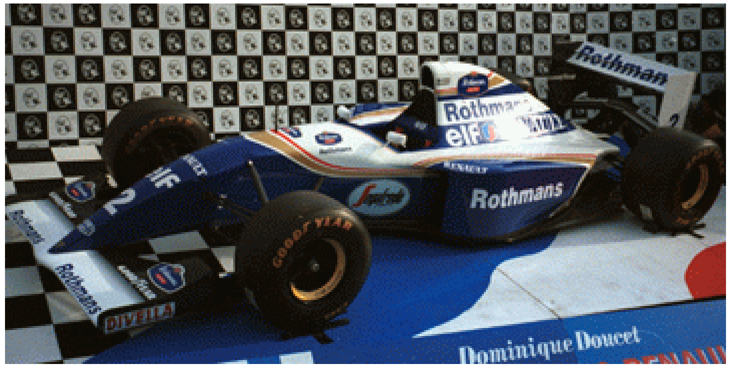
R G B	colors
0 0 0	**black**
0 0 1	**blue**
0 1 0	**green**
0 1 1	**cyan**

R G B	colors
1 0 0	**red**
1 0 1	**magenta**
1 1 0	**yellow**
1 1 1	**white**

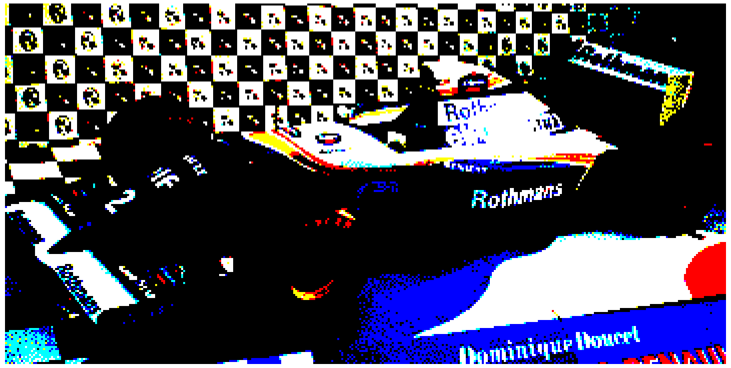
Simple Thresholding

Threshold each brightness level into 0 or 1 by comparing with 0.5

```
For each pixel
    If its brightness is less than 0.5 then output 0
    otherwise output 1
```



Input image



Output by simple thresholdiong

We cannot represent intermediate brightness well in this method.
An image of all 0.5 is not distinguishable from the one of all 1s.

Improvement of Simple Thresholding (1)

Ordered Dither

Using different thresholds for different places, instead of using the same threshold over an image.

1	9	3	11
13	5	15	7
4	12	2	10
16	8	14	6

M(i, j)

Dither matrix

$\text{if } a(i,j) < M(i,j)/16 \text{ then } b(i,j) = 0$

$\text{else } b(i,j) = 1$

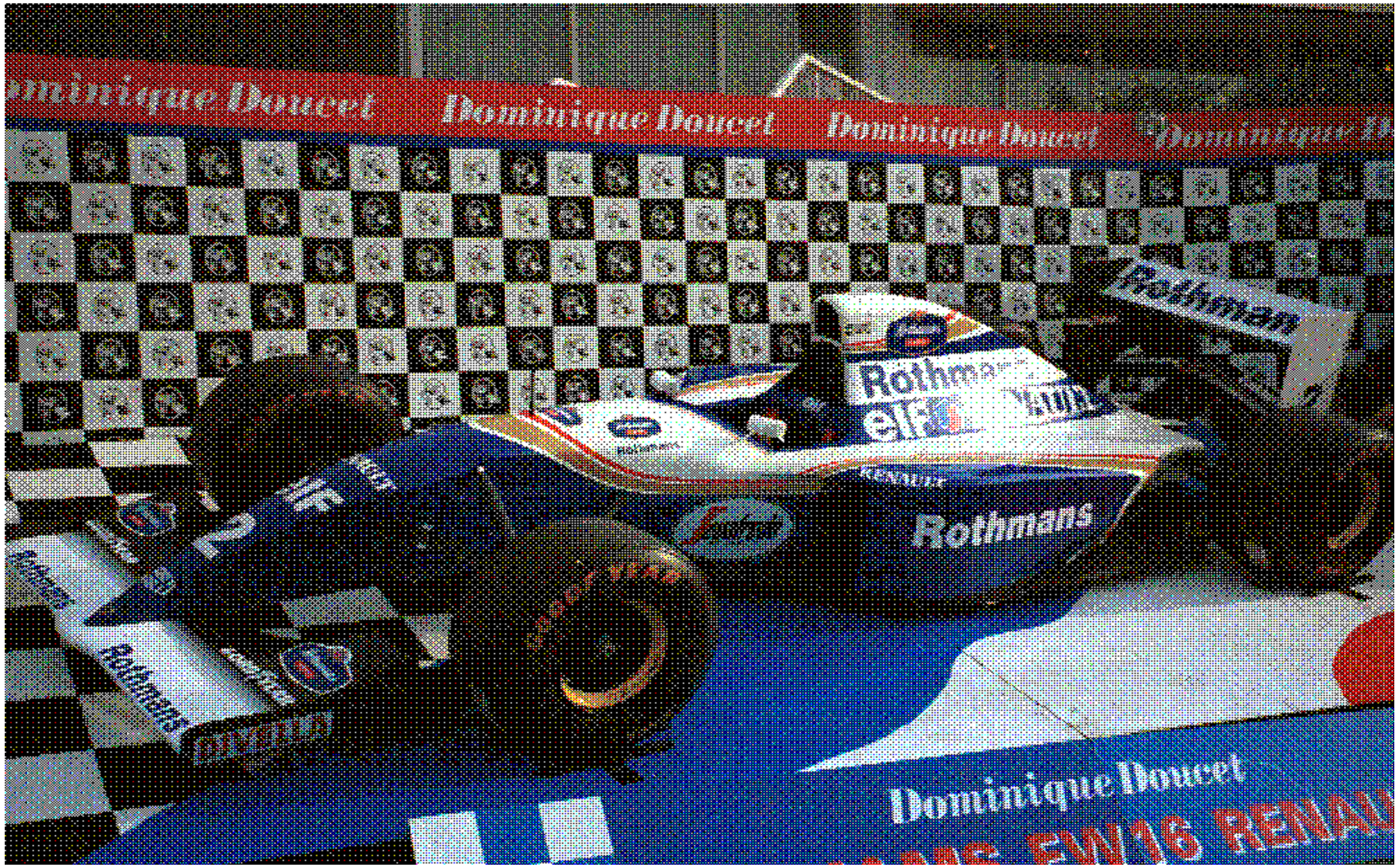
```
for i=1 to N
  for j=1 to N
      if a(i,j) < M(i%4, j%4)/16
                        then b(i,j) = 0
                        else  b(i,j) = 1
```

Larger dither matrix

	0	1	2	3	4	5	6	7
0	0	32	8	40	2	34	10	42
1	48	16	56	24	50	18	58	26
2	12	44	4	36	14	46	6	38
3	60	28	52	20	62	30	54	22
4	3	35	11	43	1	33	9	41
5	51	19	59	27	49	17	57	25
6	15	47	7	39	13	45	5	37
7	63	31	55	23	61	29	53	21

8x8 dither matrix

Output by Ordered Dither



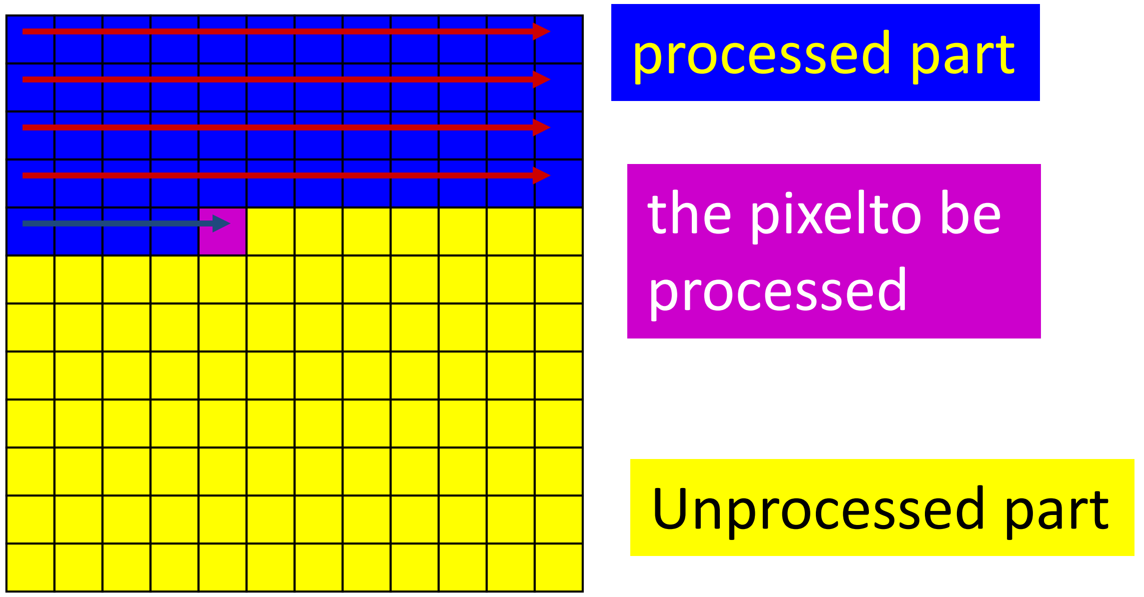
Dither matrix is recognized as a texture.

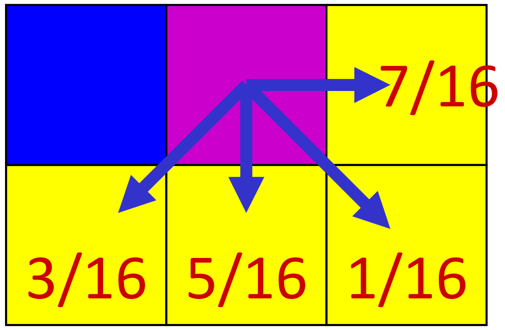
Another Improvement

(Floyd & Steinberg, 1975)

Error Diffusion Method

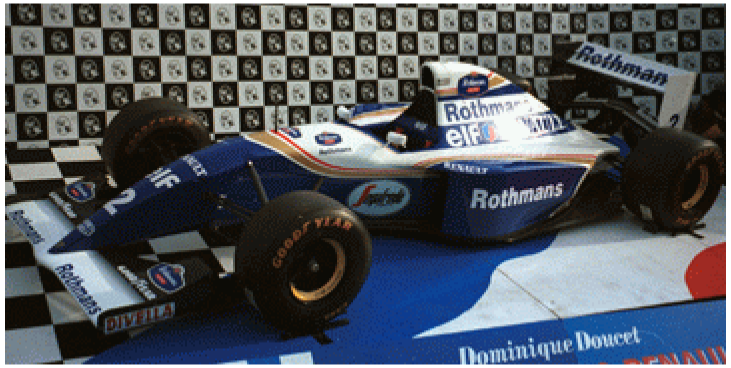
We distribute rounding errors into neighboring unprocessed pixels.



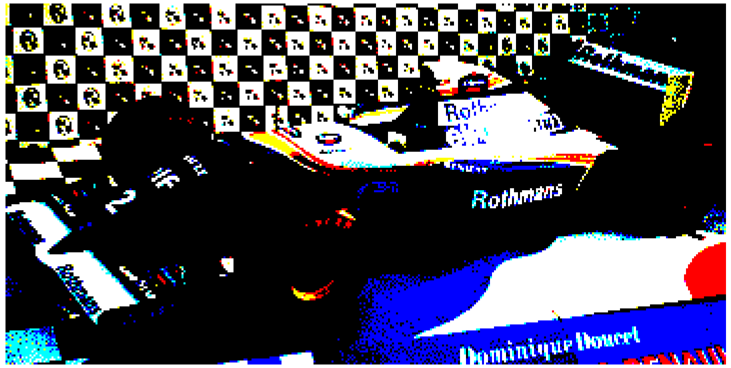


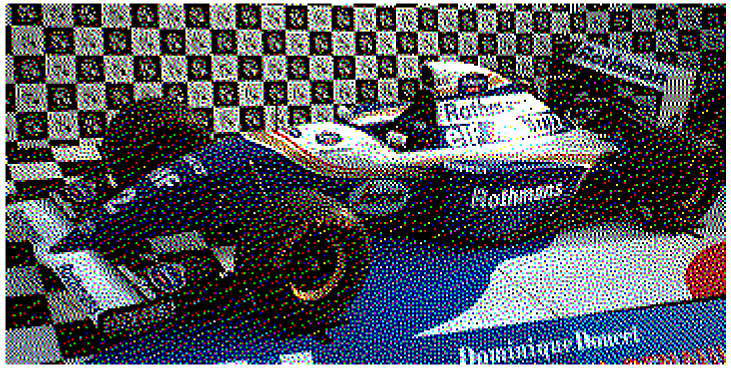
Coefficients for distributing errors

Output by Error Diffusion



Input image


Simple thresholding


Error diffusion

Larger example

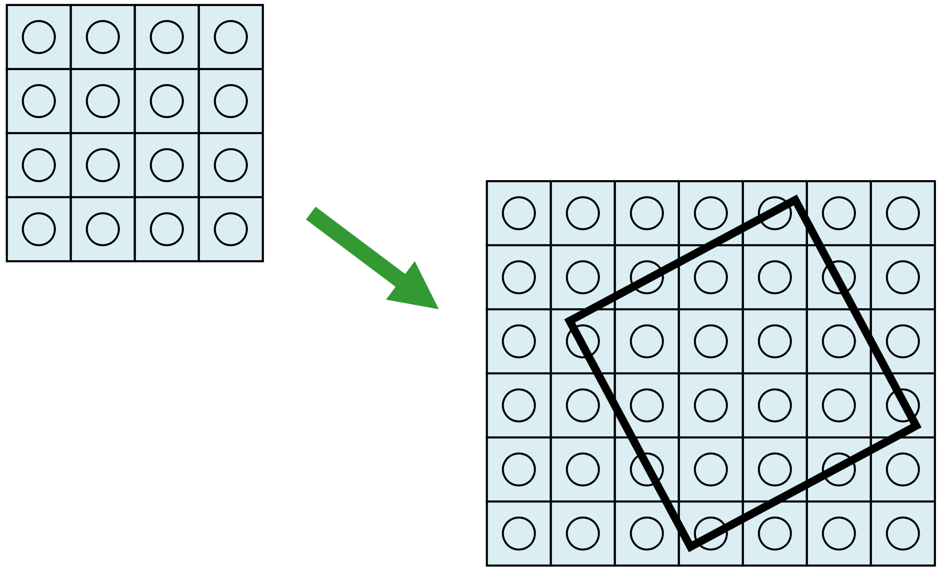
Effect of human perception

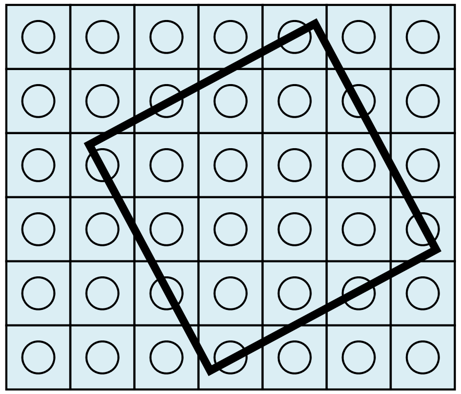
Human eyes are more sensitive to horizontal and vertical patterns

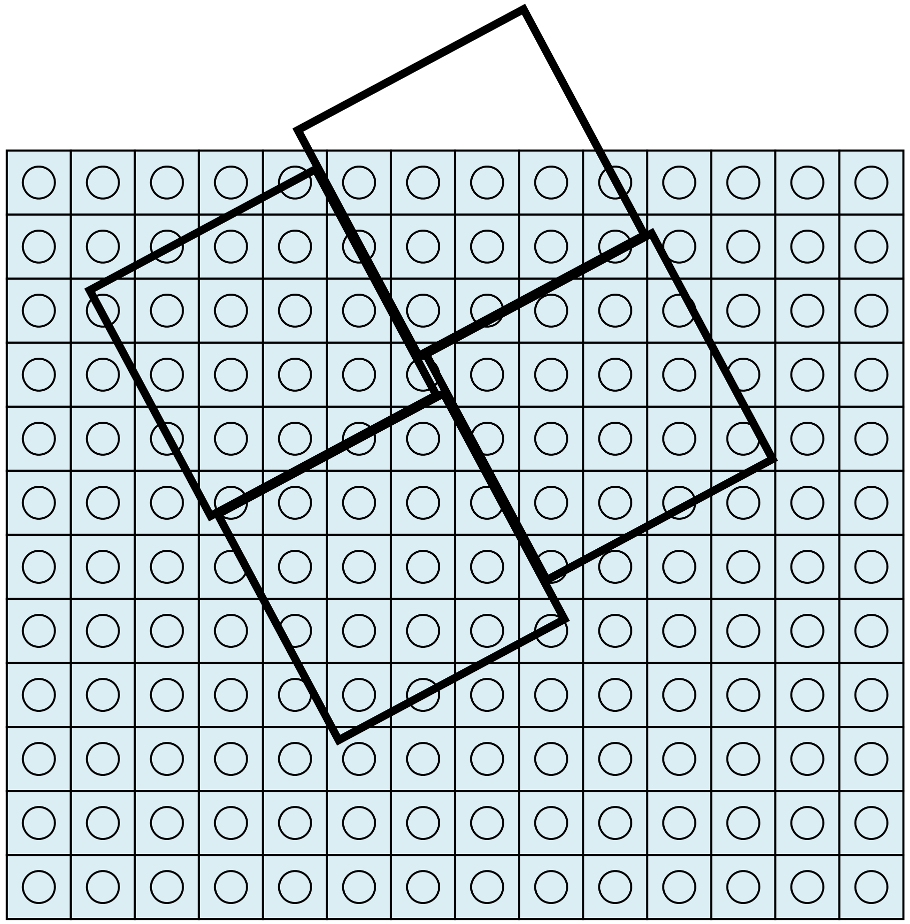
1	9	3	11
13	5	15	7
4	12	2	10
16	8	14	6

M(i, j)

Rotate this matrix by a specified angle







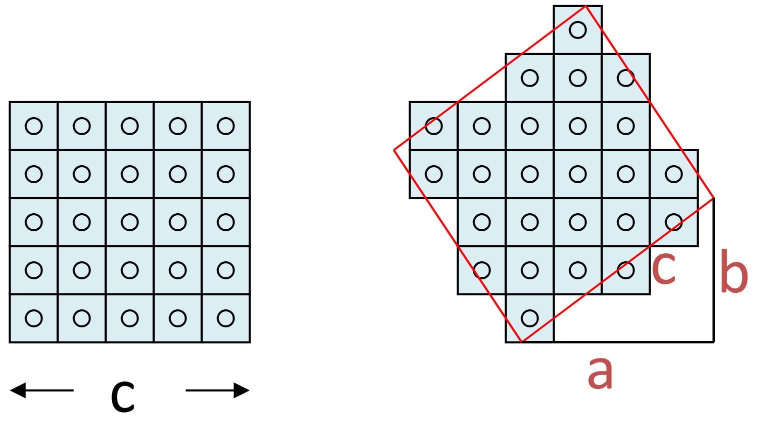
Rotated pattern should tile the entire plane.

Rotated Dispersed Dither Algorithm

V. Ostromoukhov, R.D. Hersh, and I. Amidror

SIGGRAPH'94

rotation by a Pythagorean angle



$c^2=a^2+b^2$

Pythagorean number

rotated square contains the same number of pixel centers.

one-to-one correspondence

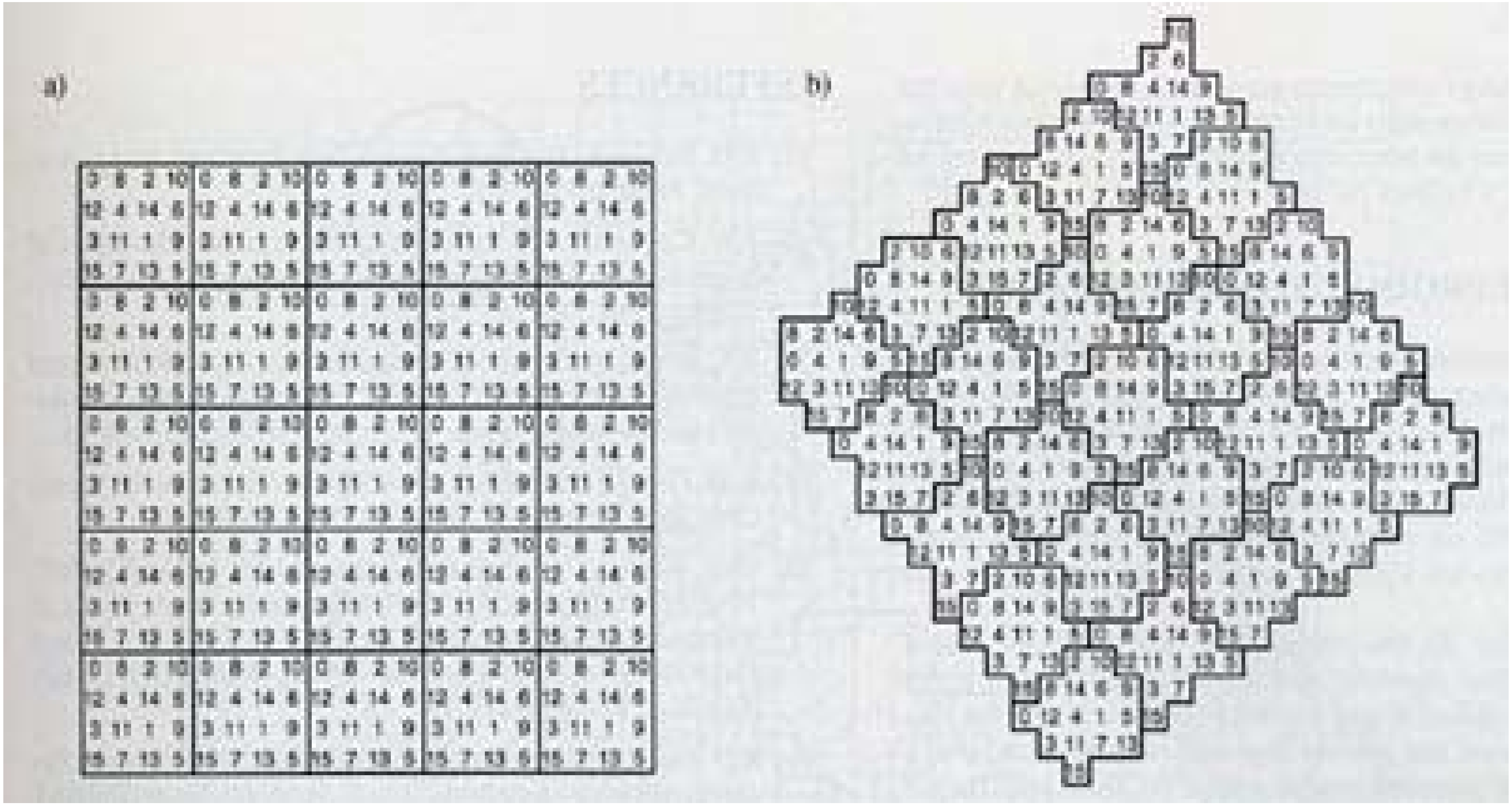
$x = \text{round}((a/c)*(i-i_0) - (b/c)*(j-j_0))+x_0$

$y = \text{round}((b/c)*(i-i_0) + (a/c)*(j-j_0))+y_0$

$\cos \rightarrow a/c, \ \sin \rightarrow b/c$

$$\begin{matrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{matrix}$$

dither matrix D^4 $\Rightarrow$ c*c array of D^4 $\Rightarrow$ ROTATE



How to find rotated pattern

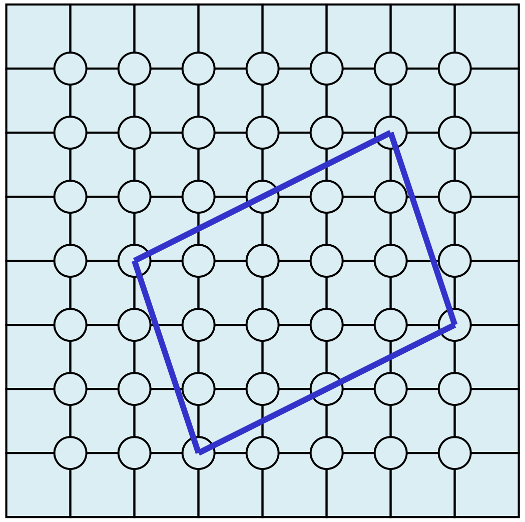
Pick's Theorem: Given a rotated pattern R, the area of R and the number of lattice points within R is related by

area(R) = #int(R) + #on(R)/2 + 1,

where area(R): area of R

#int(R): number of lattice points within R

#on(R): number of lattice points on the boundary

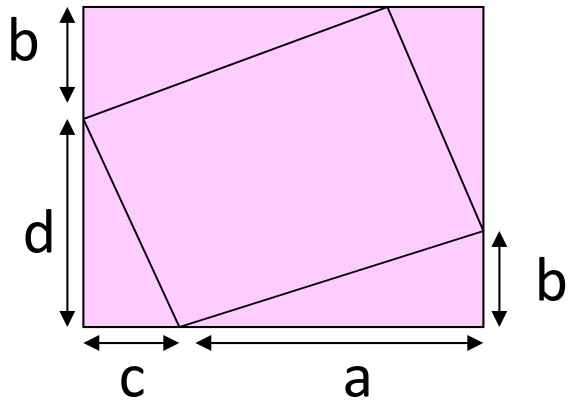


area(R)=4*3+2*1=14

#int(R)=12,

#on(R)=2,

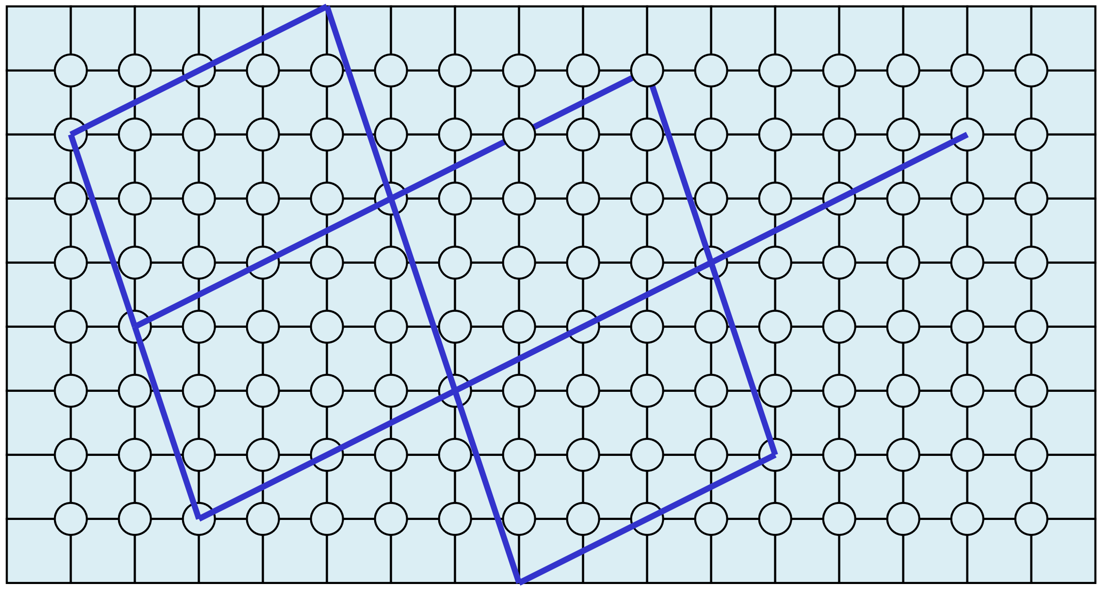
14=12 + 2/2 + 1

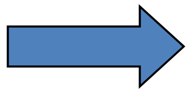


area(R)

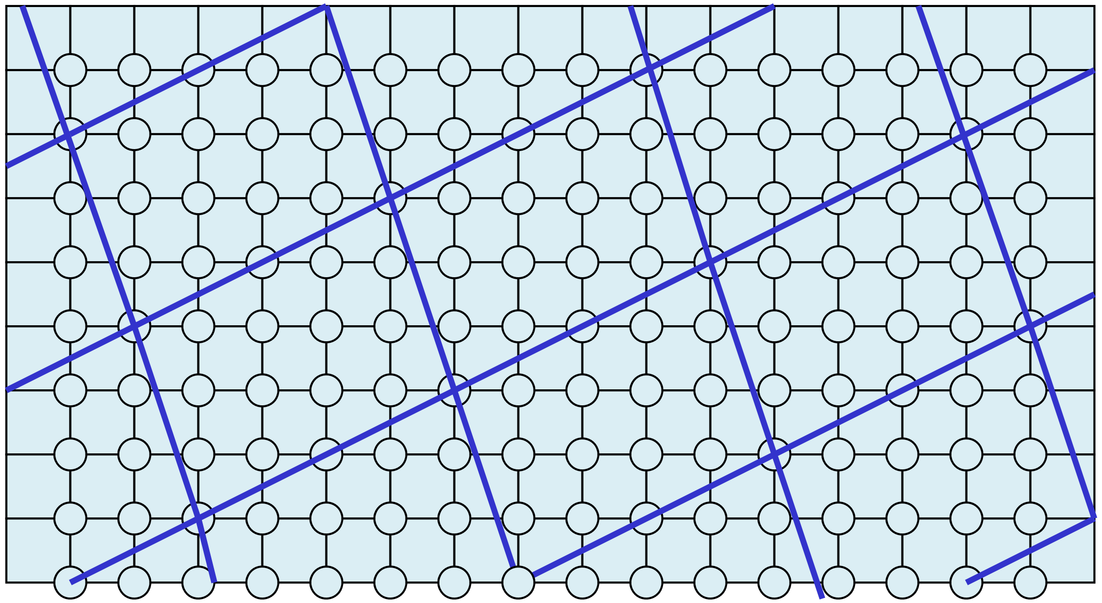
= (a+c)(b+d) - ab - cd

= ad + bc





tiling the entire plane



Problem: Given a number of lattice points *n* and an angle *t*, find a tiling pattern consisisting of *n* lattice points whose angle is roughly *t*.

Algorithm for finding a tiling pattern:

(Step 1) Enumerate all (a,b,c,d) of integers s. t.

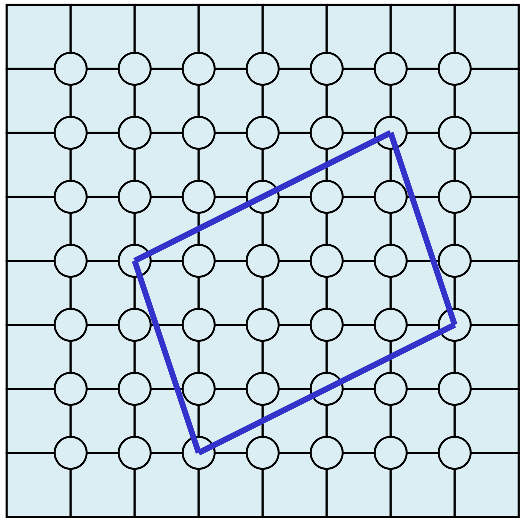
$$n = ad + bc$$

(Step 2) Choose (a,b,c,d) among them so that

$$\max\{|b/a - t|, |c/d - t|\} \rightarrow \min$$

(Step 3) Compute the tiling pattern:

- take all internal lattice points
- take the lower left corner
- take lattice points on the lower boundary (not on the upper boundary)
- take lattice points on the left boundary (not on the right boundary)



Experimental result 1

Area(R) = 64, slope = 15 degree

0-th (7,2,4,8) slope diff= 0.017765

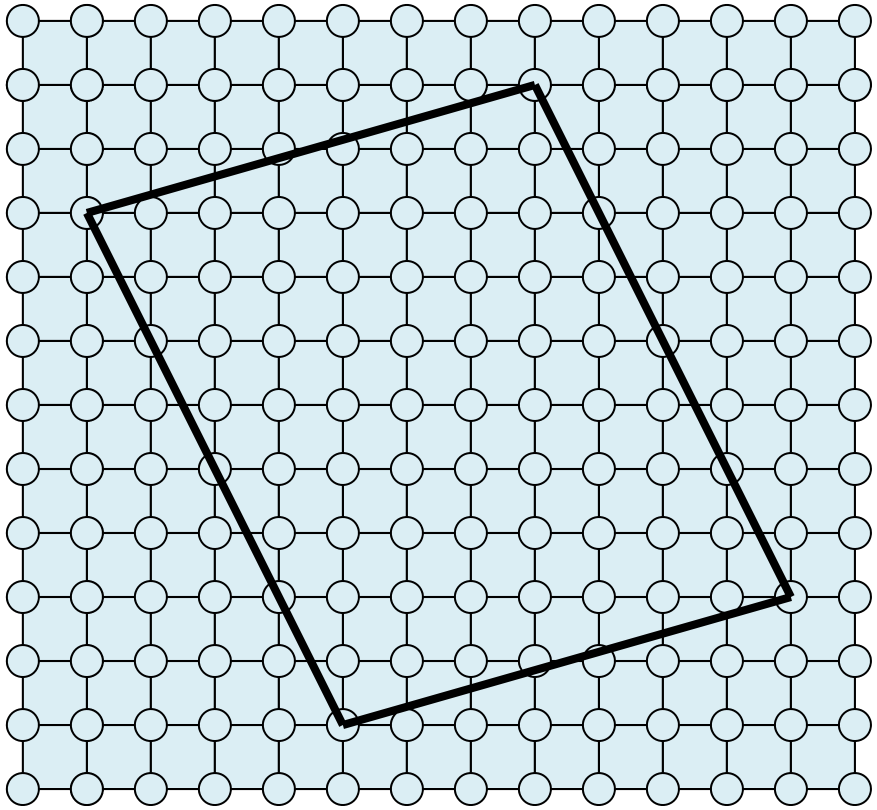
1-th (8,4,2,7) slope diff= 0.017765

2-nd (7,4,2,8) slope diff= 0.017949

3-rd (8,2,4,7) slope diff= 0.017949

4-th (7,1,1,9) slope diff= 0.125092

5-th (9,1,1,7) slope diff= 0.125092



Experimental result 2

Area(R) = 64, slope = 30 degree

0-th (6,4,4,8) slope diff= 0.077350

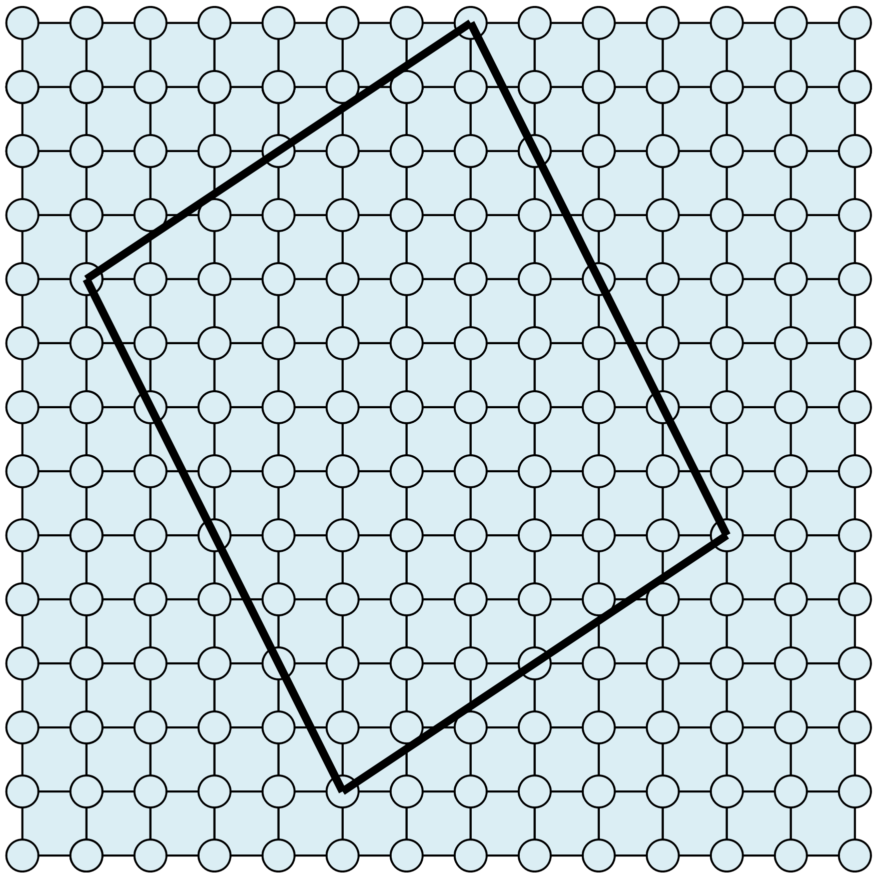
1-th (8,4,4,6) slope diff= 0.077350

2-nd (7,4,2,8) slope diff= 0.005922

3-rd (8,2,4,7) slope diff= 0.005922

4-th (7,3,5,7) slope diff= 0.136935

5-th (7,5,3,7) slope diff= 0.136935



Experimental result 3

Area(R) = 64, slope = 45 degree

0-th (7,3,5,7) slope diff= 0.285714

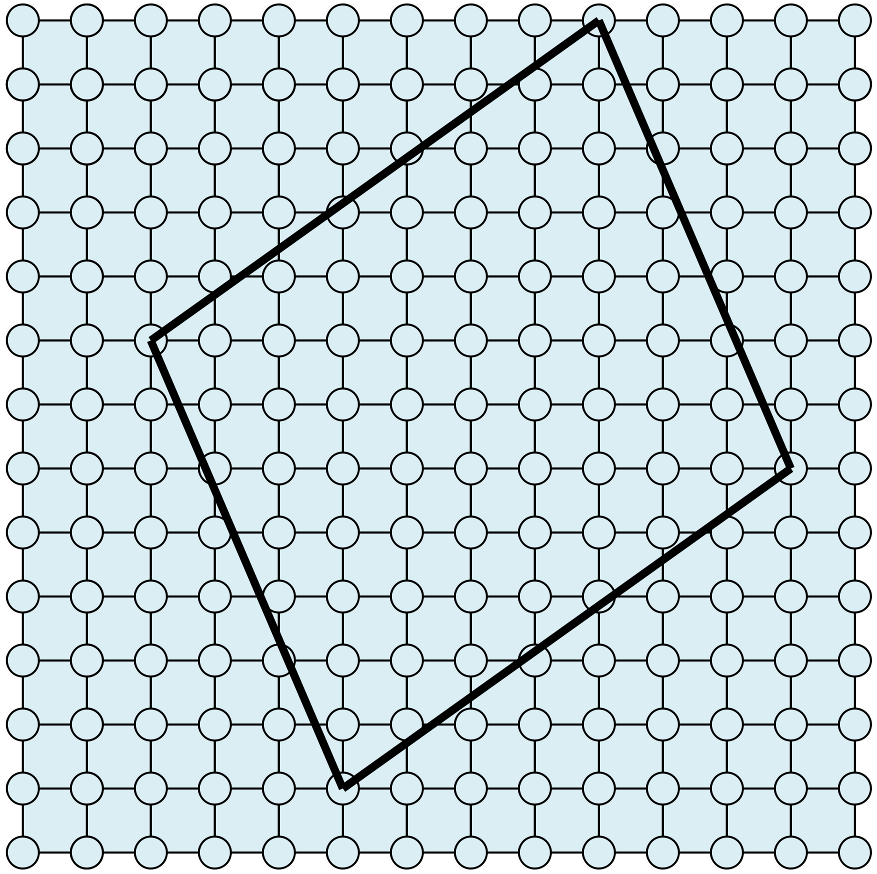
1-st (7,5,3,7) slope diff= 0.285714

2-nd (6,4,4,8) slope diff= 0.333333

3-rd (8,4,4,6) slope diff= 0.333333

4-th (7,4,2,8) slope diff= 0.428571

5-th (8,2,4,7) slope diff= 0.428571



Rotated ordered dither

1	9	3	11
13	5	15	7
4	12	2	10
16	8	14	6

M(i, j)

Dither matrix

```
for each (i,j) do
  if a(i,j) > M(i%4, j%4)/16   then b(i,j)=1
                               else b(i,j)=0
```

rotated pattern

		9	3	11		
1	13	5	12	2	15	7
	4	8	14	6	10	
			16			

Digital Halftoning as a combinatorial optimization problem

A: input image (continuous brightness level)

B: output image (binary brightness level)

dist(A, B): distance between two images A and B

A looks similar to B $\Rightarrow$ small distance

otherwise $\Rightarrow$ large distance

Problem: Given an input image A, find an output binary image B that minimizes the maximum distance to A.

Difficulty of the problem depends on how to define the distance. The distance should be sensitive to our vision.

Various criteria for dissimilarity

(1) Pointwise difference

dist(A, B) = sum of $|a(i, j) - b(i, j)|$ for every pixel (i, j)

(2) Difference in 2-by-2 region

dist(A, B) = sum of $|A_2(i, j) - B_2(i, j)|$ for every pixel (i, j)

where $A_2(i, j)$ is the sum of a(i,j), a(i+1,j), a(i,j+1), a(i+1,j+1).

(3) Difference in a family *F* of regions

dist(A, B) = sum of $|A(R) - B(R)|$ for every region R in ***F***

where A(R) is the sum in region R.

(4) Weighted difference in k-by-k region

dist(A, B) = sum of $|WA_k(i, j) - WB_k(i, j)|$

where $WA_k(i, j)$ is the weighted sum in the k-by-k region at (i,j).

weight matrix: Gaussian mask

Various criteria for dissimilarity

(1) Pointwise difference

Optimal rounding is easily obtained, but not good image

(2) Difference in 2-by-2 region

Minimizing the total error is NP-complete.
(T. Asano, T. Matsui, T. Tokuyama, 2000)

(3) Difference in a family *F* of regions

Computational complexity depends on structural property of a family of regions. ⇒ exact and approximation algorithms

(4) Weighted difference in k-by-k region

Local-search based scheme + hardware acceleration using FPGA.
Time consuming but the best quality image

(3) Difference in a family *F* of regions

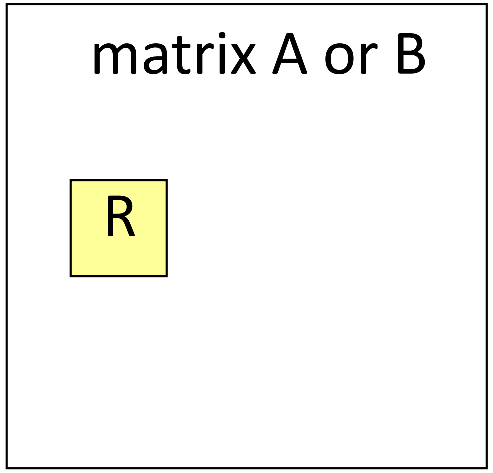
F: family of regions over the $N \times N$ grid ***G***$_n$

$$G_n = \{(1,1),(1,2),\ldots,(N,N)\} = \{p_1, p_2, \ldots, p_n\},$$
$$n = N \times N$$

the order of matrix elements is arbitrary and not fixed.

R: a region or subset of $\boldsymbol{G_n}$

$$A(R) = \sum_{p_j \in R} a_{p_j}$$



Lp-discrepancy between two matrices

$$Dist_p^F(A,B) = [\sum_{R \in F} |A(R) - B(R)|^p]^{\frac{1}{p}}$$

$$Dist_\infty^F(A,B) = \max_{R \in F} |A(R) - B(R)|$$

Problem Definition

Lp-Optimal Matrix Rounding Problem: ***$P(G_n, F, p)$***

Given a [0,1]-matrix ***A***, a family ***F*** of regions on ***G_n***, and a positive integer ***p***, find a {0,1}-matrix ***B*** that minimizes

$$Dist_p^F(A, B) = [\sum_{R \in F} |A(R) - B(R)|^p]^{\frac{1}{p}}$$

Known Results on L_∞ measure

Theorem [Baranyai, 1974]

Given a real valued matrix ***A*** and a family ***F*** of regions consisting of all rows, columns, and the whole matrix, there exists an integer-valued matrix ***B*** such that

$|\boldsymbol{A}(R) - \boldsymbol{B}(R)| < 1$ holds for every region R in ***F***.

$$\begin{array}{|ccc|c} \hline 0.4 & 0.4 & 0.7 & 1.5 \\ 0.9 & 0.3 & 0.8 & 2.0 \\ 0.4 & 0.3 & 0.5 & 1.2 \\ \hline 1.7 & 1.0 & 2.0 & 4.7 \end{array} \qquad \begin{array}{|ccc|c} \hline 0 & 1 & 1 & 1\text{-}2 \\ 1 & 0 & 1 & 2 \\ 1 & 0 & 0 & 1\text{-}2 \\ \hline 1\text{-}2 & 1 & 2 & 4\text{-}5 \end{array}$$

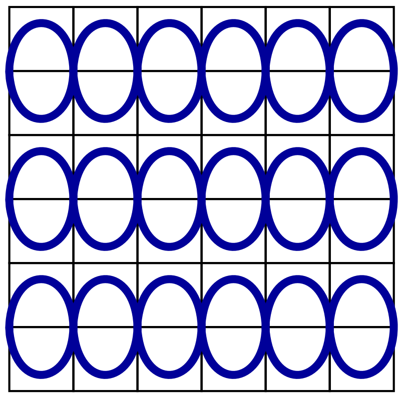
Geometric Families of Regions

family of regions is ***unimodular***
if its corresponding incidence matrix is totally unimodular

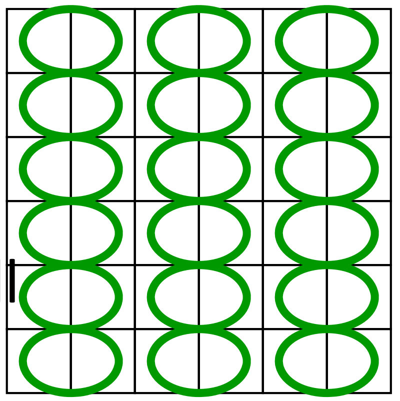
$F = \{R_1, \ldots, R_m\}$: partition family

$\Leftarrow \bigcup_{i=1}^{m} R_i = G_n$, and $R_i \cap R_j = \Phi$, for any $R_i \neq R_j$ in F

***k*-partition family** = a union of *k* different partition families



a family of all
2x1 regions



a family of all
1x2 regions

2-partition family

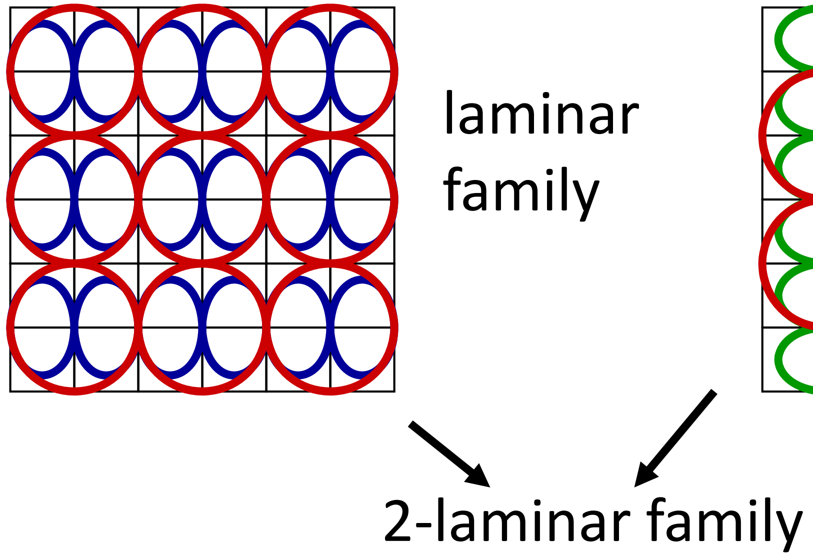
Geometric Families of Regions

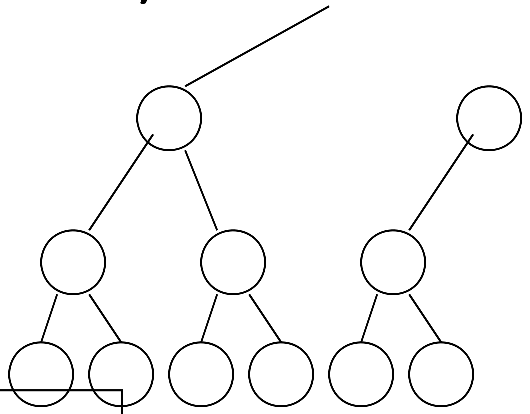
Laminar family

for any $R_i \neq R_j$ in F,

one of the followings holds :

(1) $R_i \cap R_j = \Phi$, (2) $R_i \subset R_j$, and (3) $R_j \subset R_i$





Proposition 3.1: **2-laminar family is unimodular.**

Geometric Families of Regions

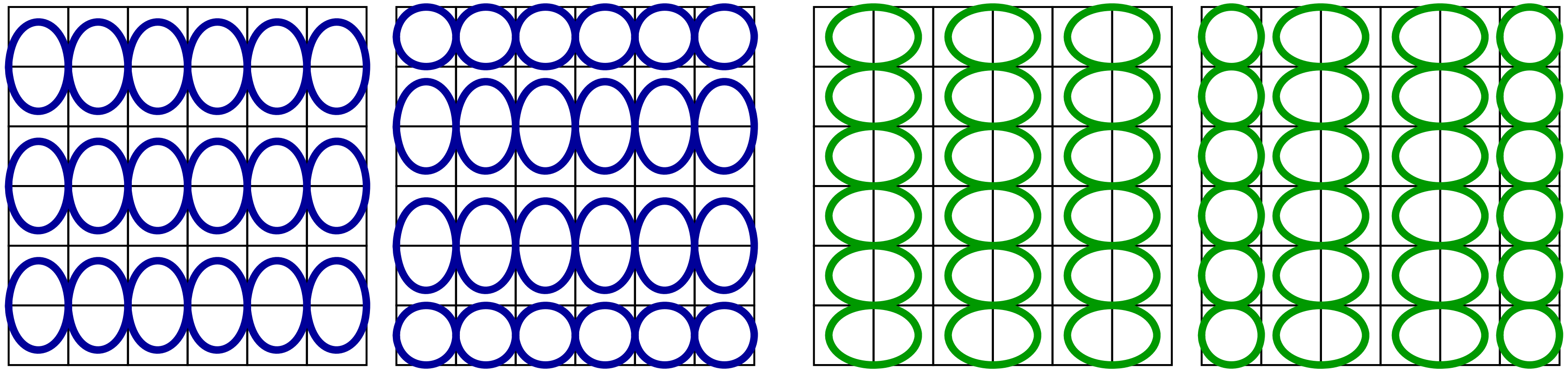
A family of all rows, all columns, and the whole matrix is 2-laminar.

$\leftarrow$ Baranyai's theorem

3-partition family is not unimodular in general.

family of all 1-by-2 and 2-by-1 regions

$\Rightarrow$ 4-partition family, but unimodular



Polynomial-time Algorithms

Theorem [Hochbaum and Shanthiskumar, 1990]
Nonlinear spearable convex optimization problem

$$\min\{\sum_{i=1}^{n} f_i(x_i) \mid Ax \geq b\}$$

on linear constraints with a **totally unimodular matrix** A can be solved in polynomial time.

⬇

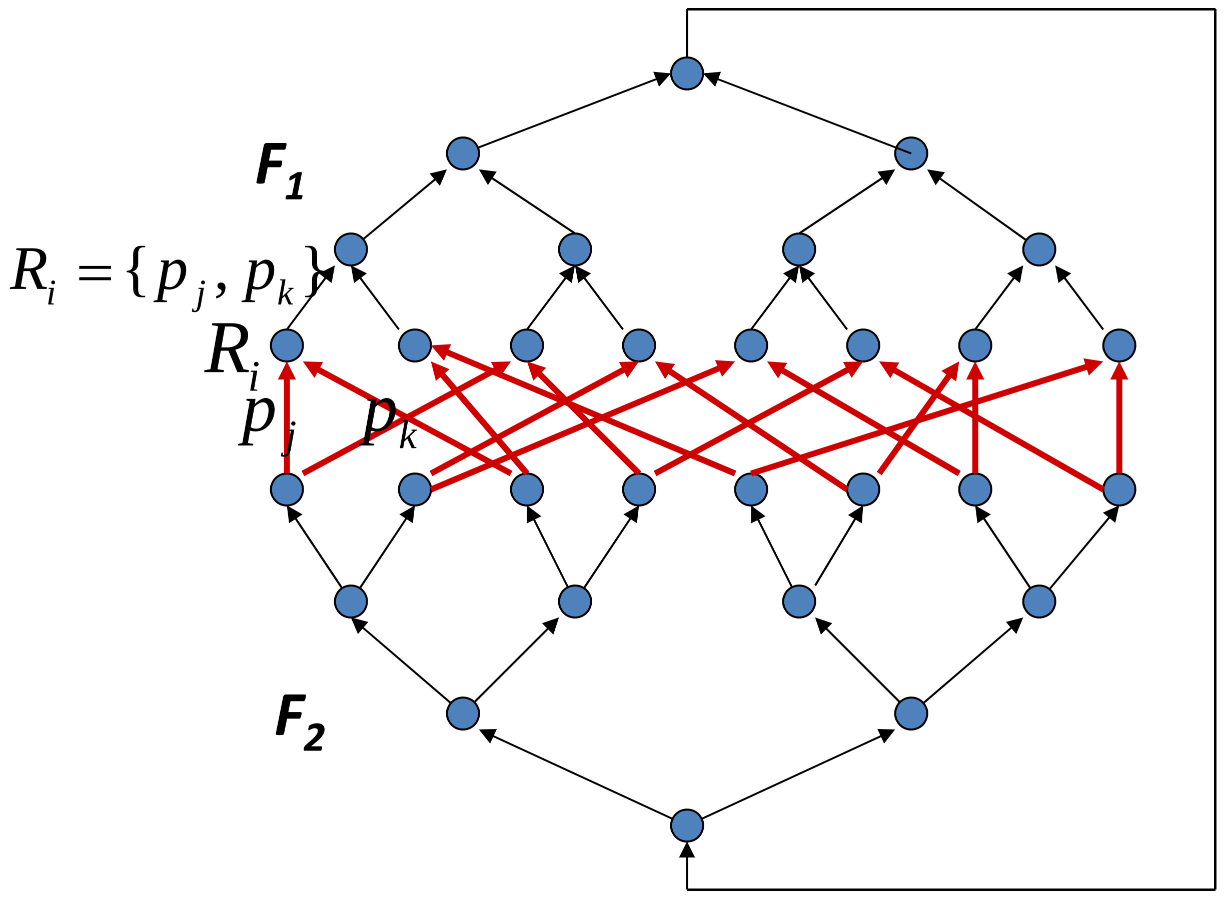
Corollary 5.2: Matrix Rounding Problem *P(*G_n*, F, p)* for *p* < ∞ is solved in polynomial time in *n*=| G_n | if its associated incidence matrix *C*(G_n, *F*) is **totally unimodular**.

need more practical algorithm

solve a matrix rounding problem for a *2-laminar family*
based on the **minimum-cost network flow algorithm**

Network-Flow type Algorithm

F: a 2-laminar family of regions
given as a union of laminar families $\boldsymbol{F_1}$ and $\boldsymbol{F_2}$



edges

$e(R_i)$ for a region

capacity : $|R_i|$

cost : $f_i(y_i)$

$e(p_j)$ for an entry

capacity : 1

cost : 0

Network-Flow type Algorithm

L1-discrepancy

Theorem 5.3: Given a [0,1]-matrix A and a 2-laminar family F, an optimal binary matrix B that minimizes the distance $Dist_1^F(A,B)$ is computed in $O(n^2 \log^2 n)$ time, where n is the number of matrix elements.

Proof: use the scaling algorithm by Edmonds and Karp

O(|E| log U (|E| + |V| log |V|)

in our case, |E|, |V|, U are all O(n).

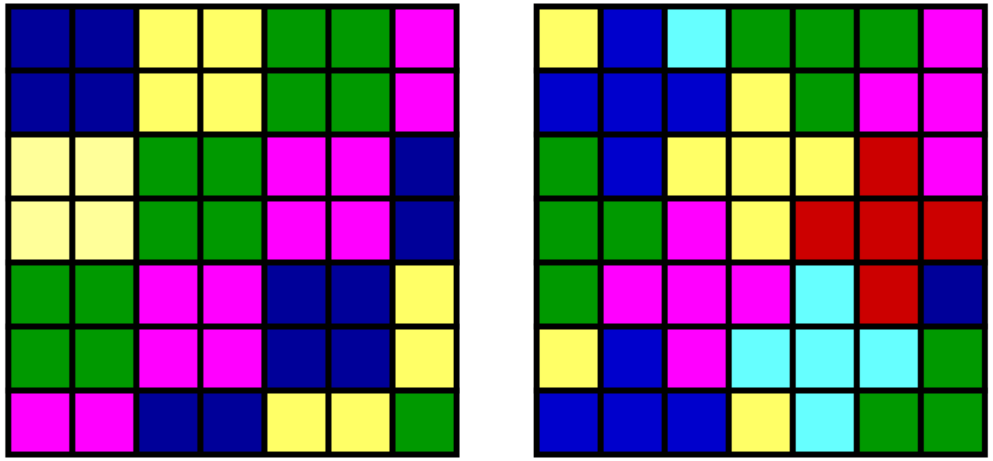
Lp-discrepancy

Theorem 5.4: Given a [0,1]-matrix A and a 2-laminar family F, an optimal binary matrix B that minimizes the distance $Dist_p^F(A,B)$ $(p \geq 2)$ is computed in $O(n^2 \log^3 n)$ time, where n is the number of matrix elements.

Proof: In this case, f_i is a piecewise-linear convex function with O(n) break points.

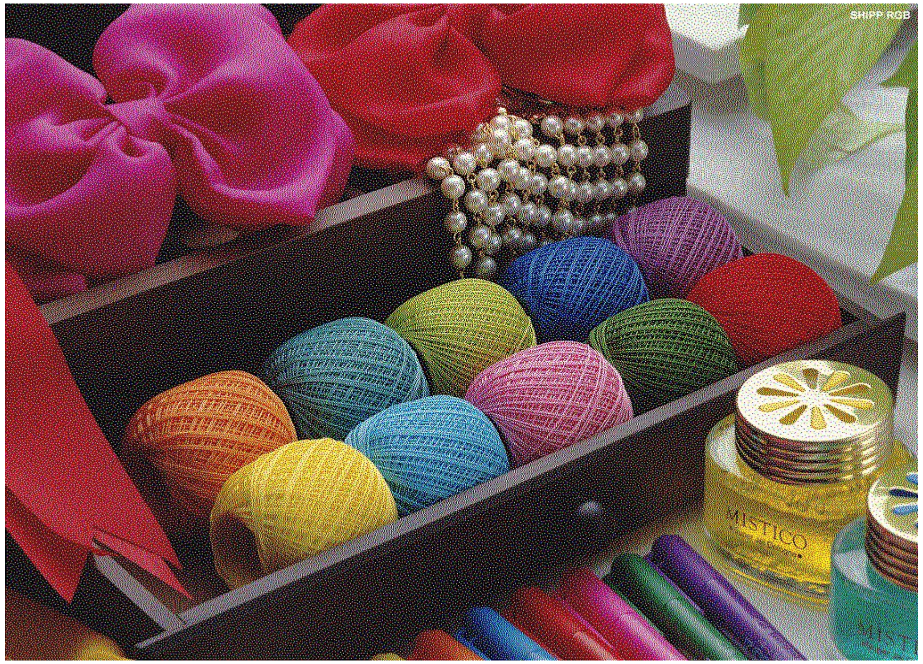
Application to Digital Halftoning

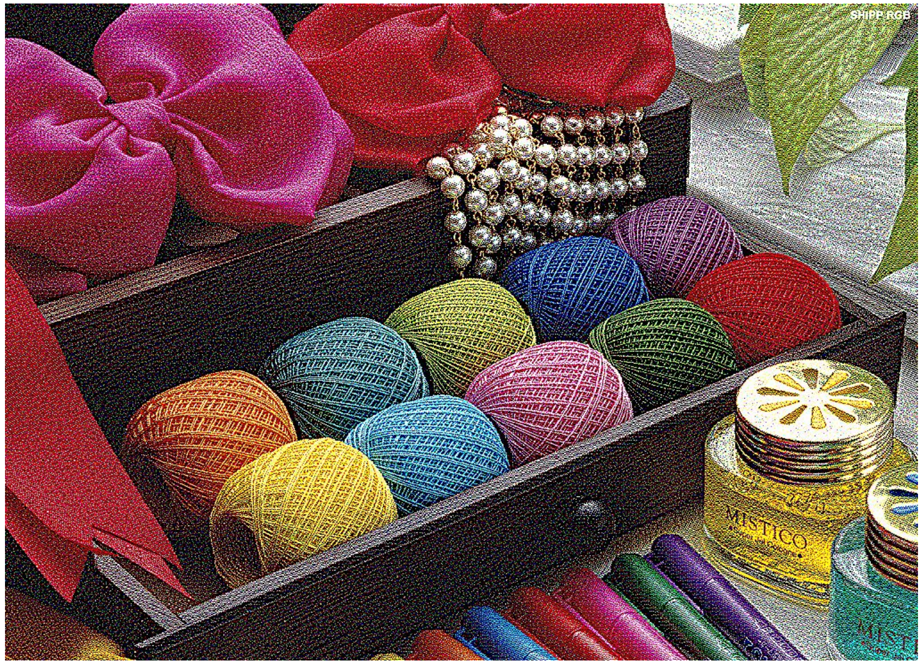
input: an intensity image *A* as a real-valued [0,1]-matrix
output: a binary image *B* consisting only of black and white dots
criterion: minimize the total sum of local error defined by |A(R)-B(R)|
for each region *R* in a 2-laminar family of regions
algorithm: minimum-cost network flow algorithm in LEDA



an example of a 2-laminar family

Error Diffusion

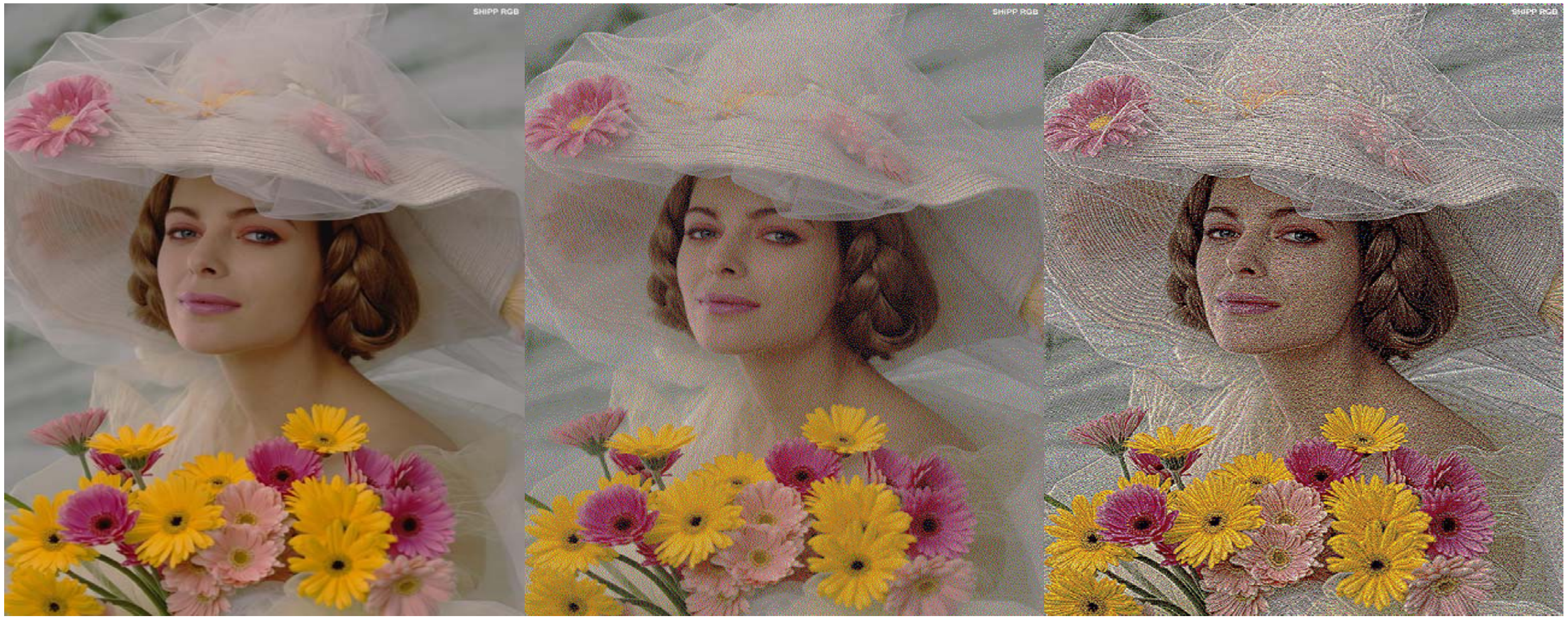




input image

MISTICO
Super Jel Cologne

MISTICO


SHIPP RGB
SHIPP RGB
SHIPP RGB

SHIPP RGB

SHIPP RGB

SHIPP RGB

SHIPP RGB

SHIPP RGB

SHIPP RGB

Recent Research Topics

Research on Memory Efficient Algorithms

to study algorithms using small amount of memory

Why do we need memory-efficient algorithm?

Data size has become very big recently than before.

1 peta byte =10^{15}

Too large to be included in main memory

Integer conversion

Define the next integer of a positive integer x

next(x) = x / 2 if x is even

next(x) = 3*x+1 if x is odd.

Example:

3->10->5->16->8->4->2->1

7->22->11->34->17->52->26->13->40->20->10->5
->16->8->4->2=>1

9->28->14->7->...

15->46->23->70->35->106->53->160->80->40->20->10->5->...

17->52->26->13->...

Collatz Conjecture: For any positive integer x, if we repeat x = next(x) then we eventually get 1 after finite iteration.

What about different next functions?

function next(x)
next(x) = x / 2 if x is even
next(x) = 5*x+1 if x is odd.

Example:

1 -> 6 -> 3 -> 16 -> 8 -> 4 -> 2 -> 1
5 -> 26 -> 13 -> 66 -> 33 -> 166 -> 83 -> 416 -> 208 -> 104 -> 52 -> 26
13->66->33->166->83->416->208->204->52->26->13
7 -> 36 -> 18 -> 9 -> 46 -> 23 -> 116 -> 58 -> 29 -> 146 -> 73 -> 366 -> 183 ->
916 -> 458 -> 229 -> 1146 -> 573 -> 2866 -> 1433 -> 7166 -> 3583 -> 17916
-> 8958 -> 4479 ->

-> 22396 -> 11198 -> 5599 -> 27996 -> 13998 -> 6999 -> 34996 -> 17498 -> 8749 -> 43746 -> 21873 -> 109366 -> 54683 -> 273416 -> 136708 -> 68354 -> 34177 -> 170886 -> 85443 ->
427216 -> 213608 -> 106804 -> 53402 -> 26701 -> 133506 -> 66753 -> 333766 -> 166883 -> 834416 -> 417208 -> 208604 -> 104302 -> 52151 -> 260756 -> 130378 -> 65189 -> 325946 ->
162973 -> 814866 -> 407433 -> 2037166 -> 1018583 -> 5092916 -> 2546458 -> 1273229 -> 6366146 -> 3183073 -> 15915366 -> 7957683 -> 39788416 -> 19894208 -> 9947104 ->
4973552 -> 2486776 -> 1243388 -> 621694 -> 310847 -> 1554236 -> 777118 -> 388559 -> 1942796 -> 971398 -> 485699 -> 2428496 -> 1214248 -> 607124 -> 303562 -> 151781 -> 758906 ->
379453 -> 1897266 -> 948633 -> 4743166 -> 2371583 -> 11857916 -> 5928958 -> 2964479 -> 14822396 -> 7411198 -> 3705599 -> 18527996 -> 9263998 -> 4631999 -> 23159996 ->
11579998 -> 5789999 -> 28949996 -> 14474998 -> 7237499 -> 36187496 -> 18093748 -> 9046874 -> 4523437 -> 22617186 -> 11308593 -> 56542966 -> 28271483 -> 141357416 ->
70678708 -> 35339354 -> 17669677 -> 88348386 -> 44174193 -> 220870966 -> 110435483 -> 552177416 -> 276088708 -> 138044354 -> 69022177 -> 345110886 -> 172555443 ->
862777216 -> 431388608 -> 215694304 -> 107847152 -> 53923576 -> 26961788 -> 13480894 -> 6740447 -> 33702236 -> 16851118 -> 8425559 -> 42127796 -> 21063898 -> 10531949 ->
52659746 -> 26329873 -> 131649366 -> 65824683 -> 329123416 -> 164561708 -> 82280854 -> 41140427 -> 205702136 -> 102851068 -> 51425534 -> 25712767 -> 128563836 ->
64281918 -> 32140959 -> 160704796 -> 80352398 -> 40176199 -> 200880996 -> 100440498 -> 50220249 -> 251101246 -> 125550623 -> 627753116 -> 313876558 -> 156938279 ->
784691396 -> 392345698 -> 196172849 -> 980864246 -> 490432123 -> -1842806680 -> -921403340 -> -460701670 -> -230350835 -> -1151754174 -> -575877087 -> 1415581862 ->
707790931 -> -756012640 -> -378006320 -> -189003160 -> -94501580 -> -47250790 -> -23625395 -> -118126974 -> -59063487 -> -295317434 -> -147658717 -> -738293584 ->
-369146792 -> -184573396 -> -92286698 -> -46143349 -> -230716744 -> -115358372 -> -57679186 -> -28839593 -> -144197964 -> -72098982 -> -36049491 -> -180247454 -> -90123727
-> -450618634 -> -225309317 -> -1126546584 -> -563273292 -> -281636646 -> -140818323 -> -704091614 -> -352045807 -> -1760229034 -> -880114517 -> -105605288 -> -52802644 ->
-26401322 -> -13200661 -> -66003304 -> -33001652 -> -16500826 -> -8250413 -> -41252064 -> -20626032 -> -10313016 -> -5156508 -> -2578254 -> -1289127 -> -6445634 -> -3222817 ->
....

For a general function next(x)

assume that next(x) always produces a positive integer

```
repeat
    x = next(x)
until (x appeared before);
```

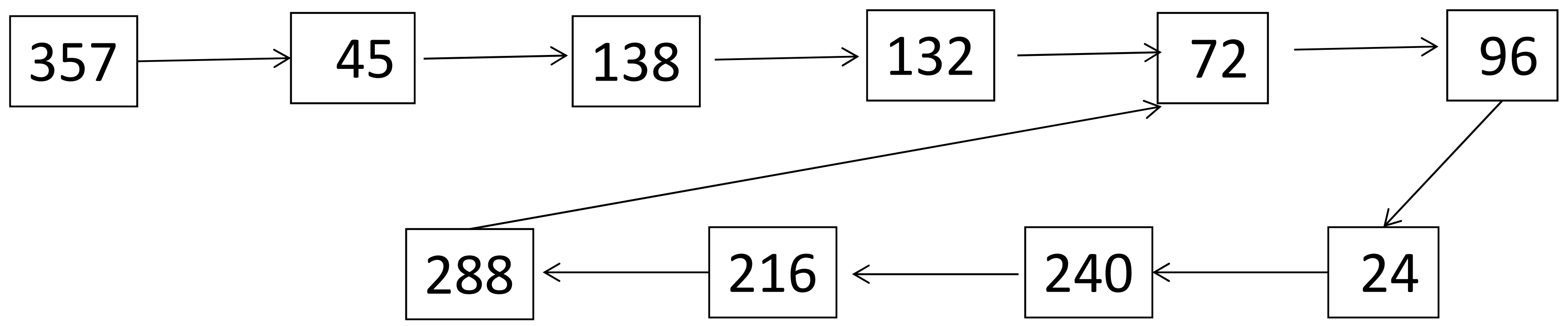
Two cases:

(1) Terminate by finding the same element

(2) Infinite sequence

Question: How can we identify a loop?

An easy solution: Memorize all integers generated.
If an integer appears twice in the output sequence, it makes a loop. But how much space do we need?



How to find a loop?

To determine an array size we need to know how long the sequence can be, but there is no way.
The sequence length may be totally independent of a starting integer x.

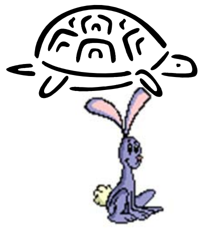
➔ Is there any way?

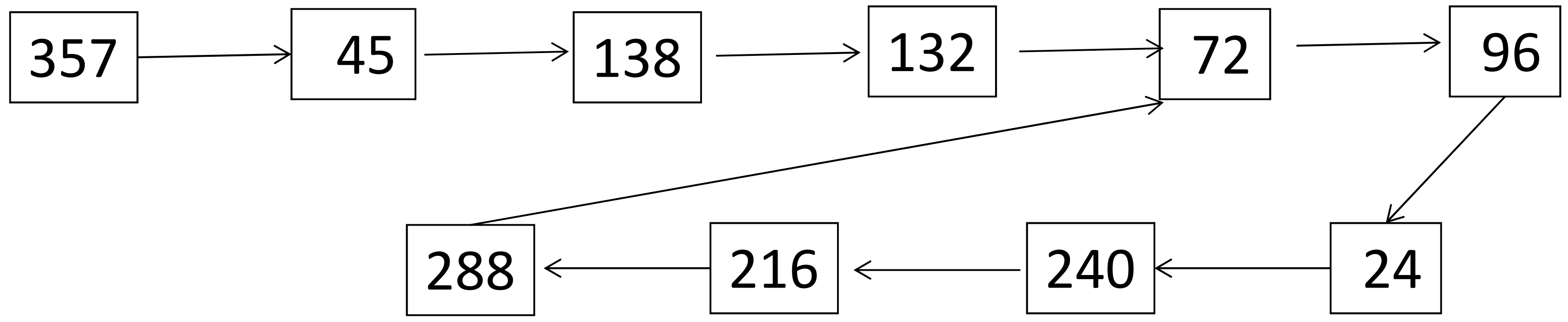
"the Tortoise and the Hare" algorithm

Two pointers of different speeds

T: one step at a time

H: two steps at a time



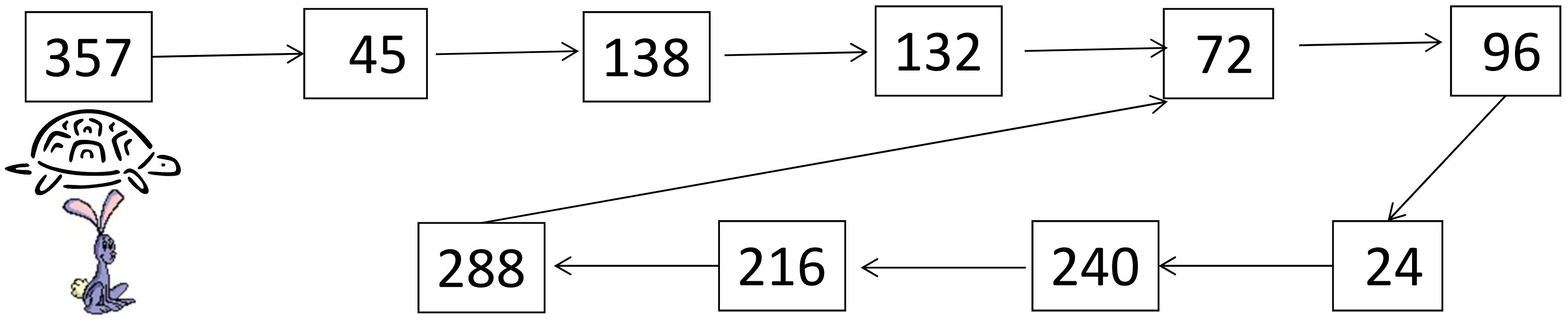


"**the Tortoise and the Hare**" algorithm

Two pointers at different speeds

T: one step at a time

H: two steps at a time



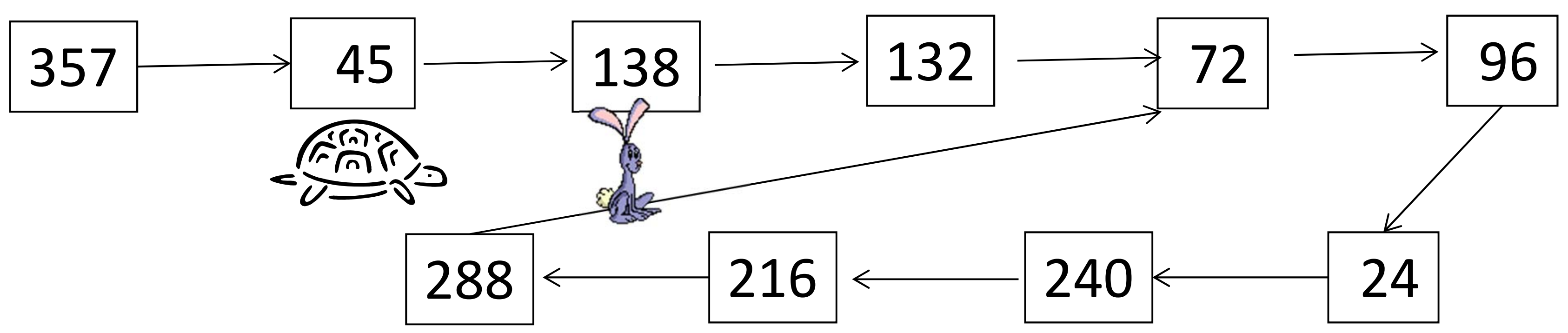
According to the Wikipedia on "Cycle detection"
D. Knuth credits the algorithm Floyd for the algorithm
in his book "The art of Computer Programming, vol. II,
Seminumerical Algorithms," Addison-Wesley.

"**the Tortoise and the Hare**" algorithm

Two pointers at different speeds

T: one step at a time

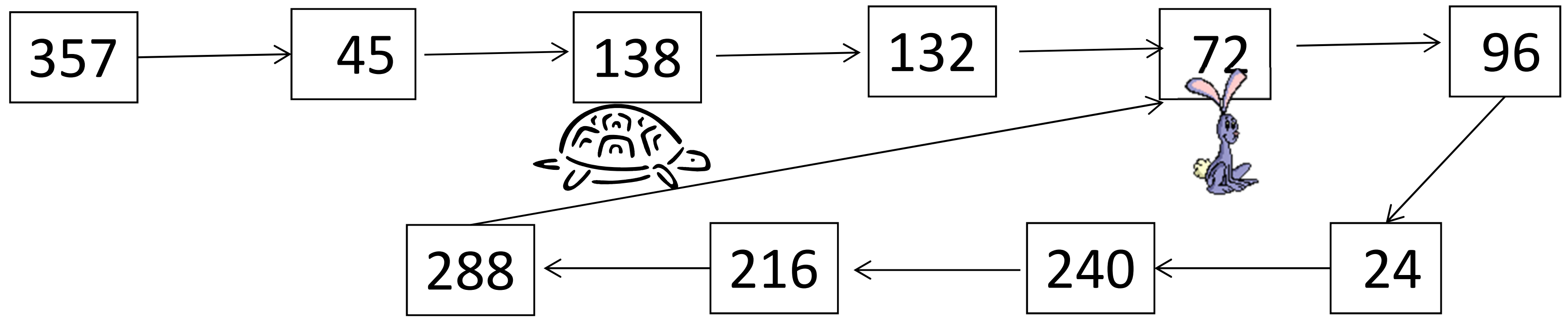
H: two steps at a time



"**the Tortoise and the Hare**" algorithm

Two pointers at different speeds

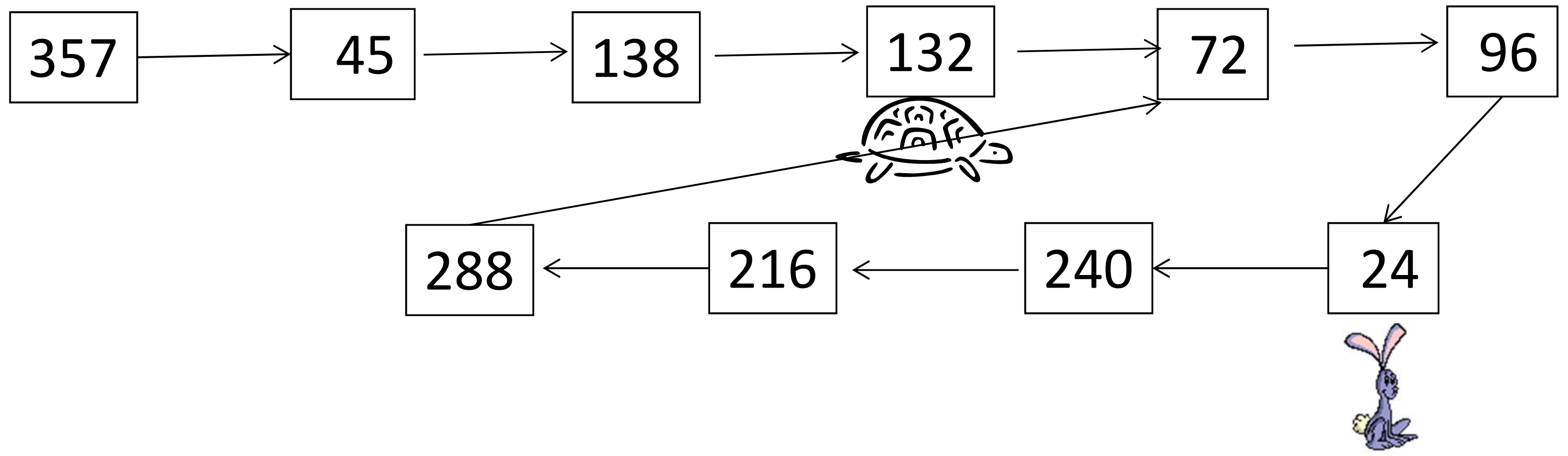
- T: one step at a time
- H: two steps at a time



"**the Tortoise and the Hare**" algorithm

Two pointers at different speeds

- T: one step at a time
- H: two steps at a time

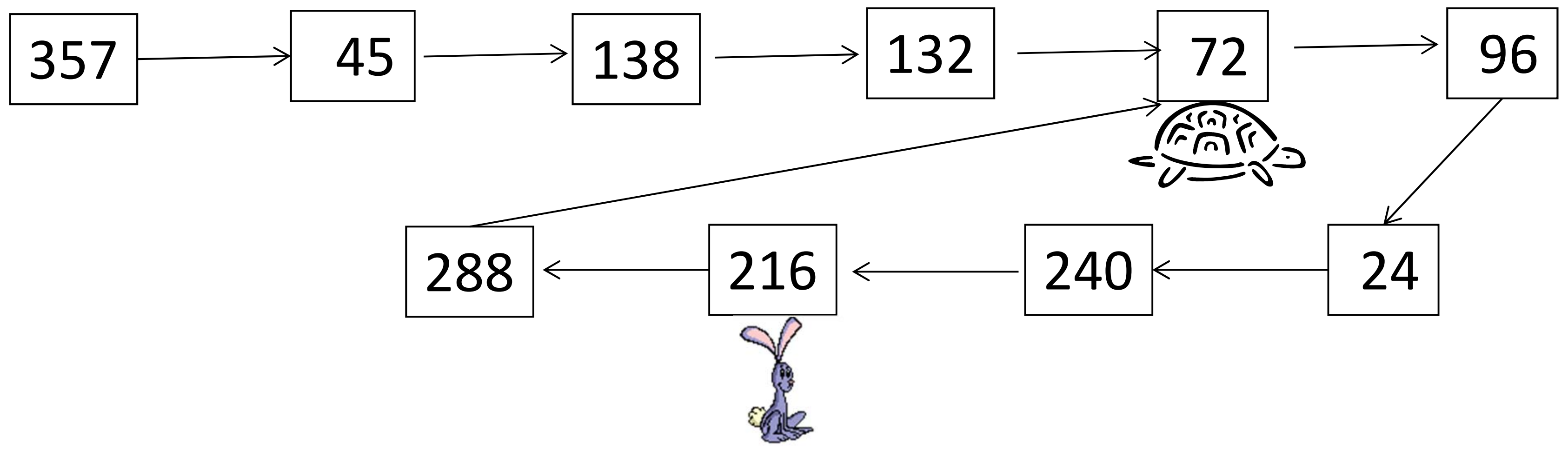


"**the Tortoise and the Hare**" algorithm

Two pointers at different speeds

T: one step at a time

H: two steps at a time

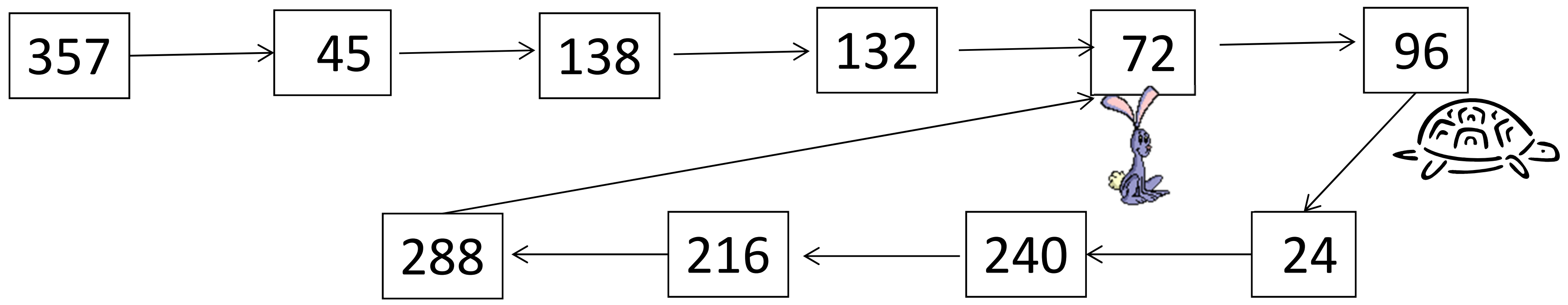


"**the Tortoise and the Hare**" algorithm

Two pointers at different speeds

T: one step at a time

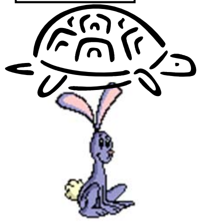
H: two steps at a time



"**the Tortoise and the Hare**" algorithm

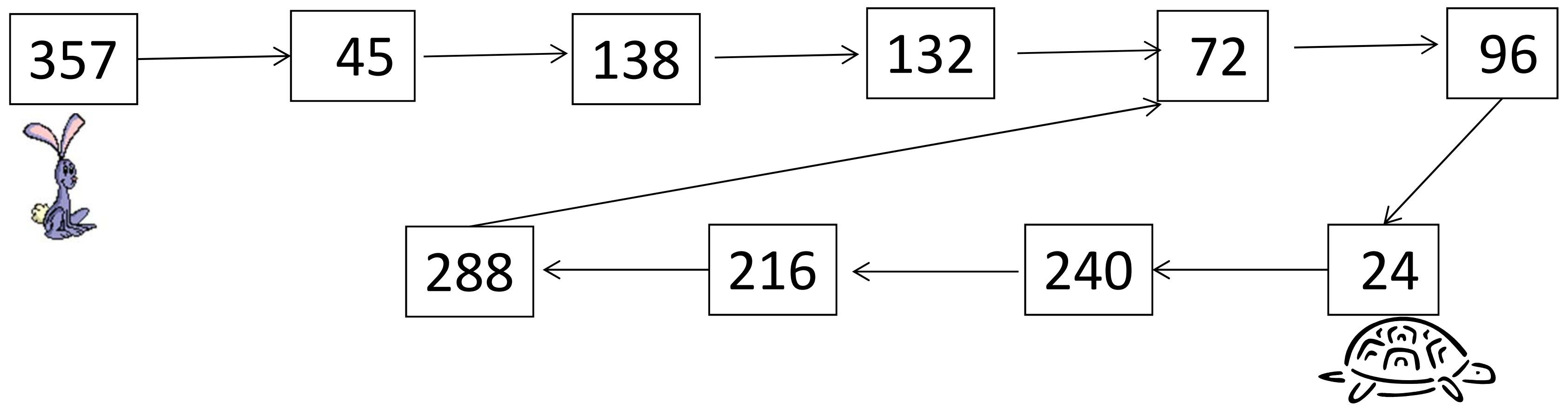
Two pointers at different speeds
- T: one step at a time
- H: two steps at a time

$$\begin{tikzcd}
\boxed{357} \arrow[r] & \boxed{45} \arrow[r] & \boxed{138} \arrow[r] & \boxed{132} \arrow[r] & \boxed{72} \arrow[r] & \boxed{96} \arrow[d] \\
 & & \boxed{288} \arrow[urr] & \boxed{216} \arrow[l] & \boxed{240} \arrow[l] & \boxed{24} \arrow[l]
\end{tikzcd}$$



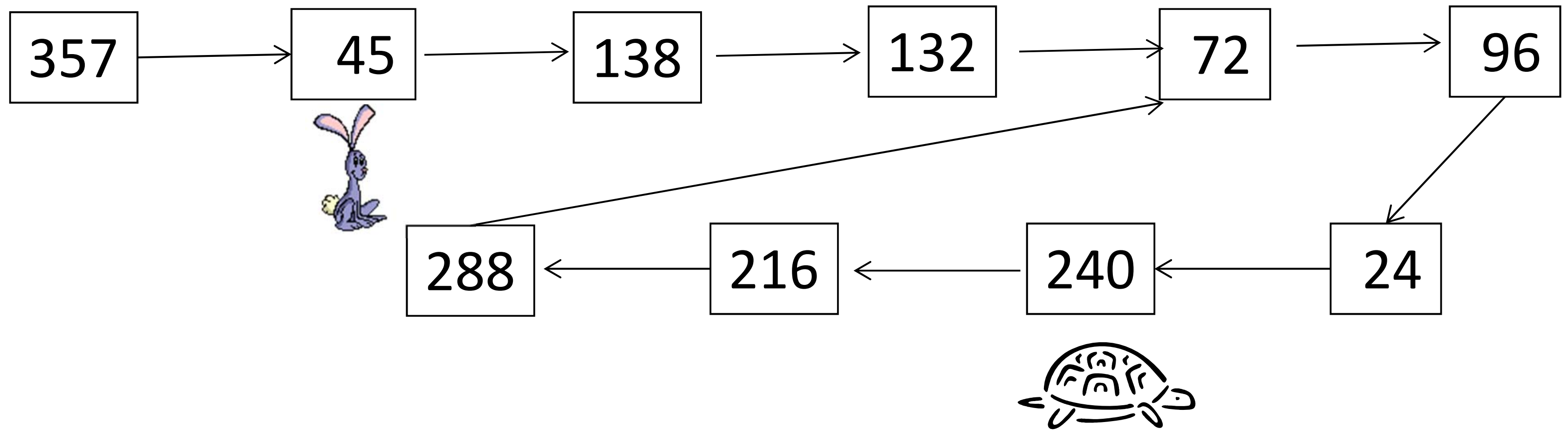
The Tortoise meets the Hare!

"the Tortoise and the Hare" algorithm



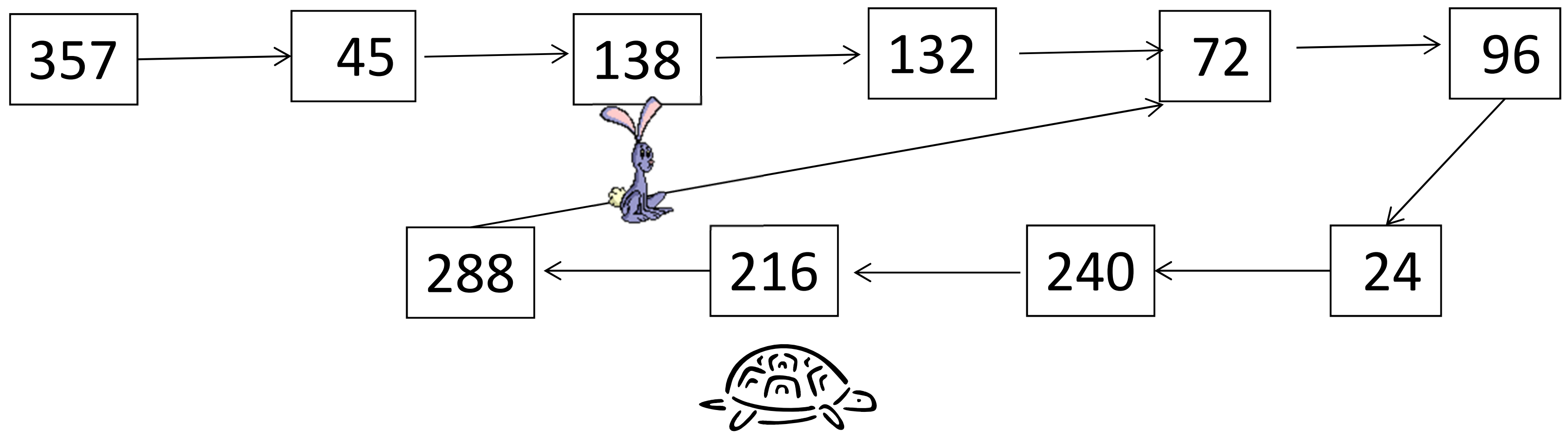
The Hare goes back to the start.

"**the Tortoise and the Hare**" algorithm



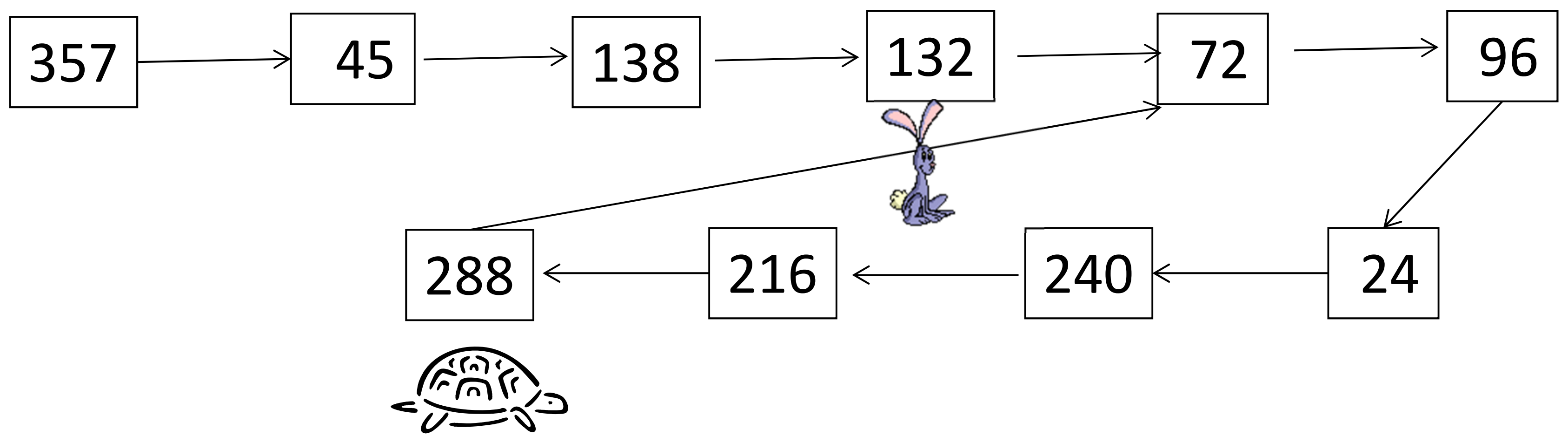
Both of them at the same speed.

"**the Tortoise and the Hare**" algorithm



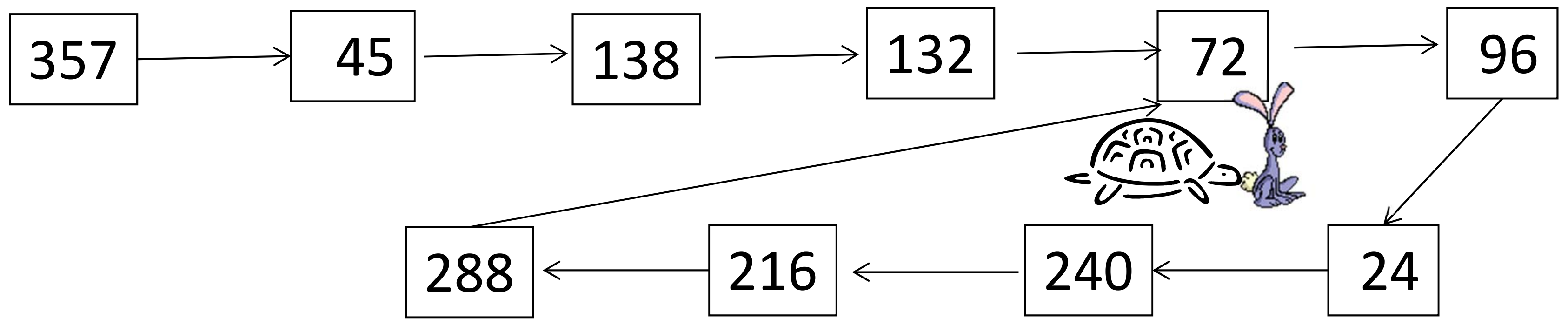
Both of them at the same speed.

"**the Tortoise and the Hare**" algorithm



Both of them at the same speed.

"the Tortoise and the Hare" algorithm



The Tortoise meets the Hare again.

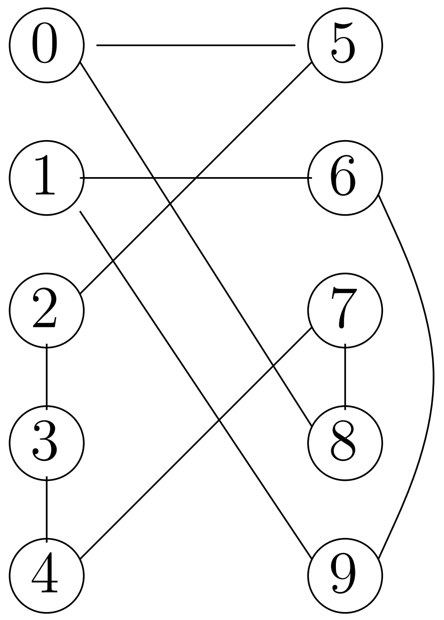
Summary

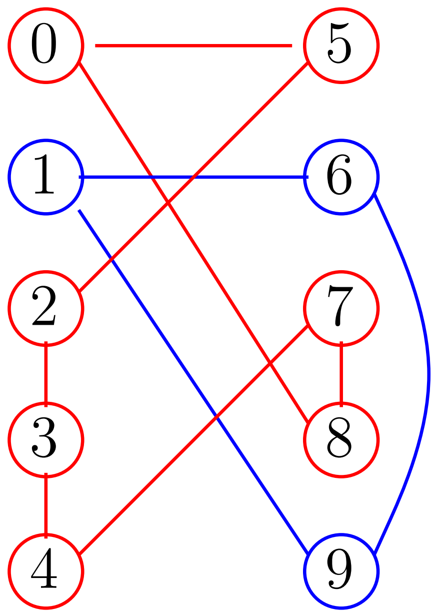
Given a function next(x) to give the next integer for a positive integer x, there is an algorithm for implementing the following algorithm in time linear in the length of sequences using only **O(1) work space**:

```
repeat
    x = next(x)
until (x appeared before);
```

Problem:
Counting the number of connected components in a 2-regular graph.

input:
0: (5, 8)
1: (6, 9)
2: (3, 5)
3: (2, 4)
4: (3, 7)
5: (0, 2)
6: (1, 9)
7: (4, 8)
8: (0, 7)
9: (1, 6)



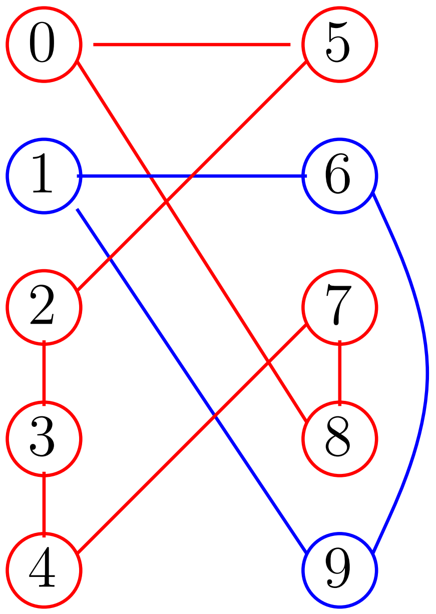


How many connected components (cycles) are there in the graph?

Algorithm with O(n) work space

```
while(there is any unmarked vertex){
    Increment the count;
    Choose an unmarked vertex u; Mark u;
    Choose any vertex incident to u as v;
    repeat{
        Mark v;
        Let {u, w} be two vertices adjacent to v;
        u = v; v = w;
    }until(v is marked)
}
```

The algorithm runs in O(n) time.



How fast constant-work space algorithm can we design?

Constant-Work-Space Algorithm for Counting the Number of Connected Components

Basic idea:

Define a **canonical vertex** in each cycle:
Each vertex has an integer index between 0 and n-1.
The vertex with the largest index in a cycle is the **canonical vertex** of the cycle.

The number of cycles = that of canonical vertices.

Algorithm for finding canonical vertices

```
for each vertex v
    follow a cycle starting from v (in one direction)
      until we encounter a vertex of a larger index or
      come back to the vertex v.
    if we come back to the vertex v
    then the vertex v is the canonical vertex in the cycle,
            and thus increment the count.
```

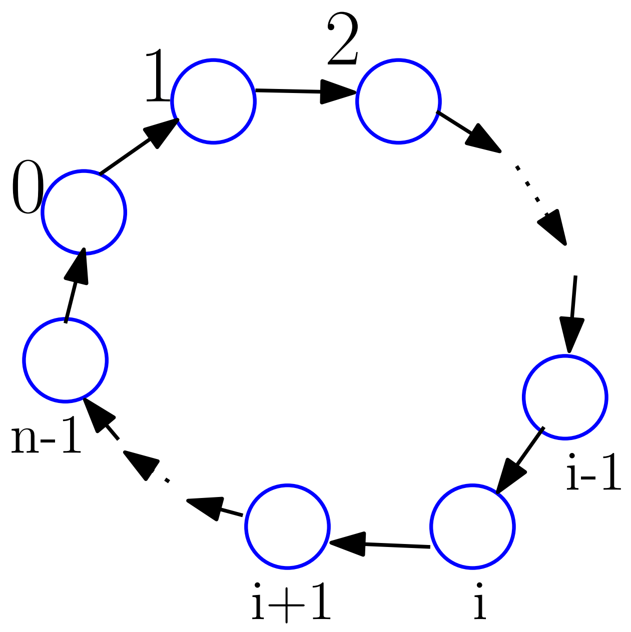
The algorithm runs in $O(n^2)$ time.

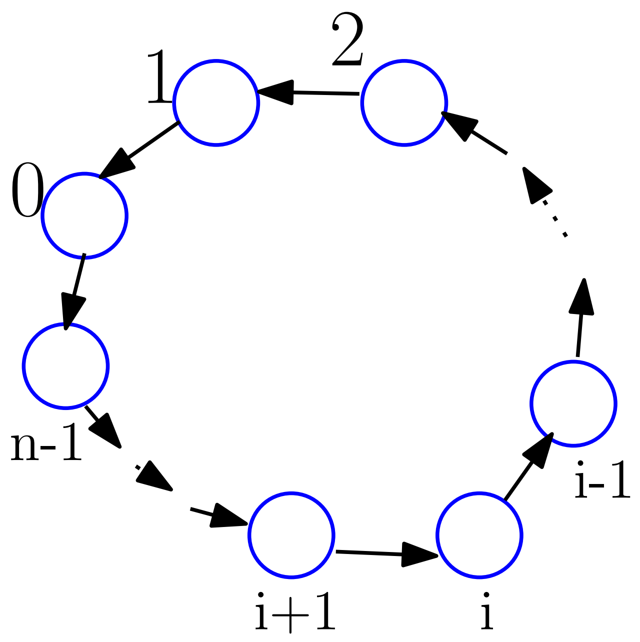
Algorithm for finding canonical vertices

```
for each vertex v
    follow a cycle starting from v (in one direction)
      until we encounter a vertex of a larger index or
      come back to the vertex v.
    if we come back to the vertex v
    then the vertex v is the canonical vertex in the cycle,
            and thus increment the count.
```

Algorithm for finding canonical vertices

```
for each vertex v
    follow a cycle starting from v (in two directions)
      until we encounter a vertex of a larger index or
      come back to the vertex v.
    if we come back to the vertex v
    then the vertex v is the canonical vertex in the cycle,
            and thus increment the count.
```





Unidirectional search takes quadratic time, but bidirectional search is mush faster.

Power of Bidirectional Search

Theorem: [A., Bereg, Kikrpatrick, 2009]

Bidirectional Search counts the number of connected components of a 2-regular graph in O(n log n) time.

Fun of Algorithm Research

- Never ending.

  Never ends since we iterate improvement.
- Everywhere possible once we have a pen and paper.

  Computers may sometimes help.
- Computers may have a great help occasionally.
- When they help, its effect is great.

Thank you very much!