Chap 9. DDA and Bresenham's Algorithms  Sat 01/03/25

DDA - Digital Differential Analyzer.
- methods for line drawing in computer graphics.

DDA - calculates intermediate points using slope.

Ex: DDA Algorithm.

Step 1. Input 2 end points $(x_1, y_1)$ and $(x_2, y_2)$

Step 2. Calculate $dx = x_2 - x_1$ and $dy = y_2 - y_1$

Step 3. Obtain nb of steps by using conditions:

If abs(dx) > abs(dy) then steps
  steps = abs(dx)
else
  Steps = abs(dy)

Step 4. ~~Decrement~~ Determine the increment values of x and y as
  X inc = dx / steps and
  Y inc = dy / steps

Step 5. plot starting point (x, y)

Step 6. for k = 1 to steps in increments of 1

Step 7. plot next point by generating it as
  x = x + xinc
  y = y + yinc

Step 8. End for

Step 9. stop.

Example: Given points $P_1(2, 3)$ and $P_2(7, 5)$ calculate all ~~points~~ intermediate points using DDA and then plot it on cartesian plan.

Sol:

① Input $P_1(2,3)$ and $P_2(7,5)$

② $dx = x_2 - x_1 = 7 - 2 = 5$
$dy = y_2 - y_1 = 5 - 3 = 2$ } variations

③ $[abs(dx) > abs(dy)]$ True

Hence $steps = abs(dx)$
$steps = 5$

④ Increments:

$X_{inc} = dx/steps = 5/5 = 1$

$Y_{inc} = dy/steps = 2/5 = 0.4$

⑤ Plot starting point $P_1(2,3)$

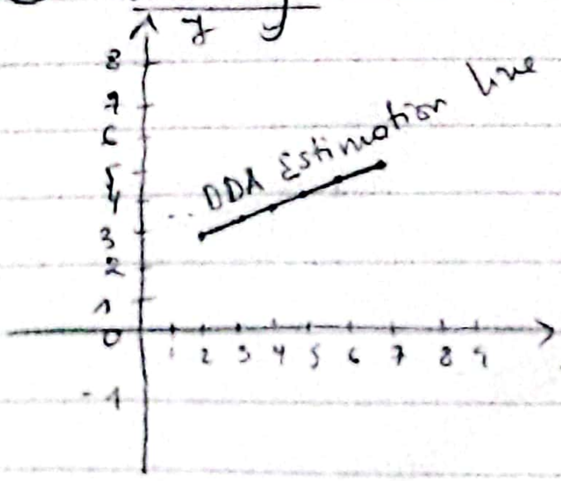
⑥ For $k = 1$ to steps:
$k = 1$ to $5$:

⑦
$x = x + X_{inc}$
$y = y + Y_{inc}$

| | | | | | | |
|---|---|---|---|---|---|---|
| x | 2 | 3 | 4 | 5 | 6 | 7 |
| y | 3 | 3.4 | 3.8 | 4.2 | 4.6 | 5 |

1 2 3 4 ... steps.

⑧ Plotting




Ex. Bresenham's Algorithm

step 1. Input 2 end points and store the left endpoints in $(x_0, y_0)$

step 2. Load $(x_0, y_0)$ into frame and plot first point

step 3. calculate $dx$, $dy$, $2dy$ and $2dy - 2dx$ and obtain the starting value for decision parameter.

as $P_0 = 2dy - dx$ // decision parameter

step 4. At each $x_k$ along the line starting at $k = 0$ check the following test:

if $P_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and

$$P_{k+1} = P_k + 2dy$$

otherwise, the next point to plot $(x_k + 1, y_k + 1)$ and $P_{k+1} = P_k + 2dy - 2dx$

step 5. Repeat step 4 dx times

Ex :

①. Input $P_1(2,3)$ and $P_2(7,5)$

②. Load $(x_0, y_0) \Rightarrow (2,3)$

③. calculate $dx$, $dy$, $2dy$ and $2dy - 2dx$

$dx = X_2 - X_1 = 5$

$dy = Y_2 - Y_1 = 2$

$2dy = 4$

$2dy - 2dx = 4 - 10 = -6$

④. $P_0 = 2dy - dx$

$= 4 - 5$

$P_0 = -1$

④. $P_0 < 0$ $(-1 < 0)$ True : $(x_k + 1, y_k)$

next point $(x_0 + 1, y_0) \Rightarrow (3,3)$

$P_{k+1} = P_k + 2dy$

$P_{0+1} = P_0 + 2dy$

$\cancel{P_1} = -1 + 4$

$P_1 = 3$

$P_1 > 0$ $(3 > 0)$ True : $x_k + 1, y_k$

Class work

Given endpoints: $(x_1, y_1) = (2, 3)$
$(x_2, y_2) = (9, 8)$

a) Apply the DDA to find the next point to draw a line from (2, 3) and (9, 8)

b) Apply Bresenham's algorithm to find next point to draw a line from (2, 3) to (9, 8)

Sol:

a) DDA

(i). Input $P_1(2,3)$ and $P_2(9,8)$

(ii). $\left.\begin{array}{l} dx = 9 - 2 = 7 \\ dy = 8 - 3 = 5 \end{array}\right\}$ variation.

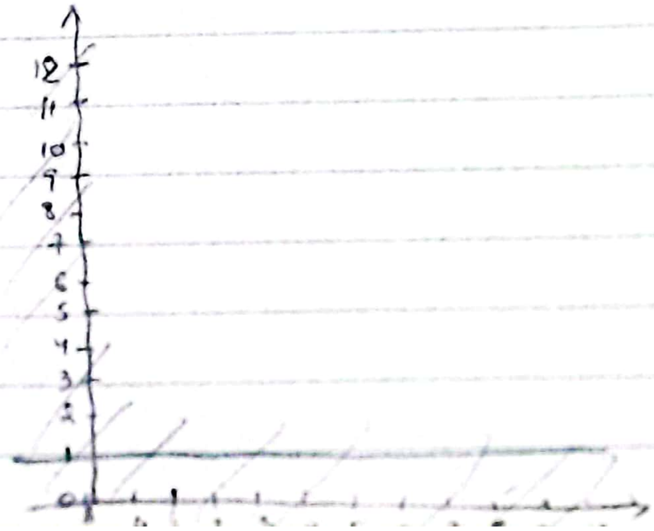
(iii). [abs (dx) > abs (dy)]
(7 > 5) True; Then
steps = abs (dx)
→ steps = 7

(iv). Increments:
X - inc = dx / steps = 7/7 = 1
Y - inc = dy / steps = 5/7 = 0.7142 ≃ 0.7

(v). For k = 1 To steps = 7:
X = X + X - inc
Y = Y + Y - inc

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| x | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| y | 3 | 3.7 | 4.4 | 5.1 | 5.8 | 6.5 | 7.2 | 8 |

(vi). Plot and End.



(b) Bresenham's Algorithm

(1). Input $P_1(2,3)$ , $P_2(9,8)$

(2). load $(x_0, y_0)$

$(2,3)$

(3). calculate $dx$, $dy$, $2dy$, $2dy - 2dx$

$P_0 = 2dy - dx$

$dx = 9 - 2 = 7$

$dy = 8 - 3 = 5$

$2dy - 2dx = 10 - 14 = -4$

$2dy = 10$

$P_0 = 2dy - dx = 10 - 7 = 3$

(4). For $k = 0$ to $dx = 7$ : $(0, 1, 2, 3, 4, 5, 6)$

* $(P_k < 0)$ false
* $(P_k > 0)$ ⇒ $k = 0$, $P_0 = 3$ $(P_0 > 0)$

point1 $(3, 4)$

$P_1 = P_0 + 2dy - 2dx$

$= 3 + (-4)$

$P_1 = -1$

* $k = 1$, $(P_1 < 0)$ True $P_1 = -1$

point 2 $= (4, 4)$

$P_2 = P_1 + 2dy$

$= -1 + 10$

$P_2 = 9$

* $k = 2$, $(P_2 > 0)$.

point 3 $(5, 5)$

$P_3 = P_2 + 2dy - 2dx$

$= 9 + (-4)$

$P_3 = 5$

* $k = 3$, $(P_3 > 0)$

point 4 $(6, 6)$

$P_4 = P_3 + 2dy - 2dx$

$= 5 + (-4)$

$P_4 = 1$

CamScanner

* $k = 4$, ($P_k > 0$)

point 5 (7,7)

$P_5 = P_4 + 2dy - 2dx$

$= 1 + (-4)$

$P_5 = -3$

* $k = 5$, ($P_k < 0$) $P_5 = -3$

point 6 (8,7)

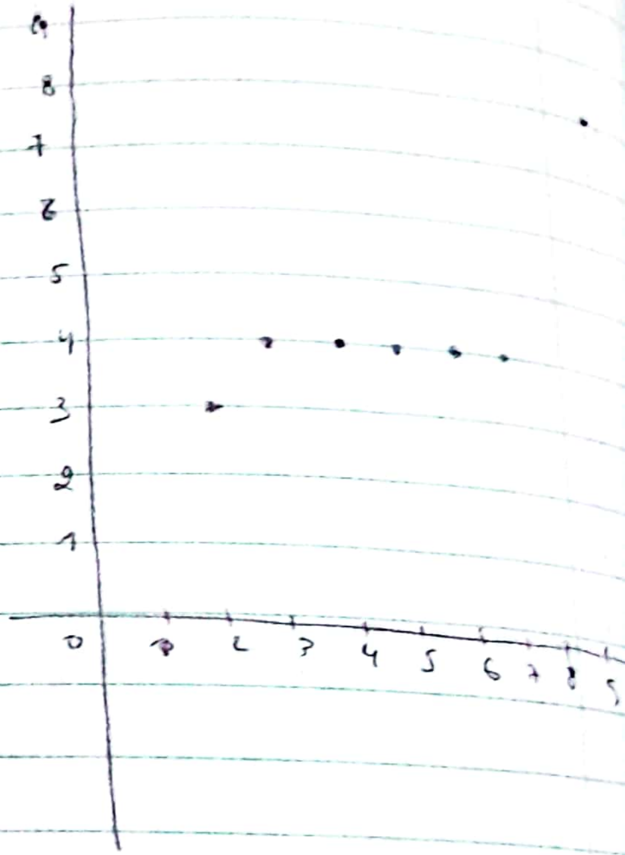
$P_6 = P_5 + 2dy$

$= -3 + 10$

$P_6 = 7$

* $k = 6$ ($P_k > 0$)

point 7 (9,8)




⑤ Table of points

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| x | 2 | 3 | 4 | 5 | 6 | 7 | 9 |
| y | 3 | 4 | 4 | 5 | 6 | 7 | 8 |

* $k = 4$, $(P_k > 0)$

point 5 $(7, 7)$

$P_5 = P_4 + 2dy - 2dx$

$= 1 + (-4)$

$P_5 = -3$

* $k = 5$, $(P_k < 0)$ $P_5 = -3$

point 6 $(8, 7)$

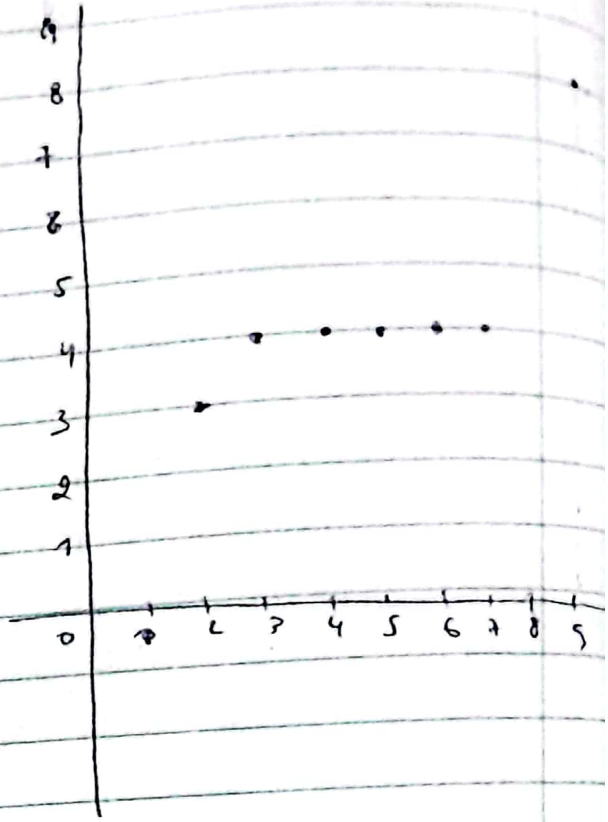
$P_6 = P_5 + 2dy$

$= -3 + 10$

$P_6 = 7$

* $k = 6$ $(P_k > 0)$

point 7 $(9, 8)$



⑤. Table of points

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| x | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| y | 3 | 4 | 4 | 5 | 6 | 7 | 7 | 8 |

# Banker's Algorithm

↳ algorithm used in OS to ensures that the system does not enter an unsafe state where processes may **deadlock** due to conflicting resource requests.

It's primarily used in systems where multiple processes are competing for a finite number of resources.

⚠ **Summary of Bresenham's (Rewind)**

① Input $P_1$, $P_2$

②. load initial point $(x_0, y_0)$

③. calculate: $dx$, $dy$, $2dy$, $2dy - 2dx$, $P_0 = 2dy - dx$ (decision parameter)

4. for $k = 0$ to $dx$ do:

    if $P_k > 0$:

    newpoint $(x_k + 1, y_k)$

    $P_{k+1} = P_k + 2dy$

    else:

    newpoint $(x_k + 1, y_k + 1)$

    $P_{k+1} = P_k + 2dy - 2dx$

5. End for
6. End.

↳ 4 types of data structures

DS used to implement Banker's are: ~~typ~~

- available
- Max
- Allocation
- Need (request) ⇒ need <= available
    ↳ Max - Allocation

1. Available
    → nb of available instances of each resource in the system at a given point of ~~type~~ time.
2. Max
3. Allocation
4. need.

Example

Considering a system of five processes $P_0$ thro' $P_4$ and three resources A, B, C.

Resource type A has 10 instances

B has 5 instances

C has 7 instances.

Suppose at time $t_0$ following snapshot of the system has been taken:

Q1. What will be the content of the Need matrix?

Δ Safe state of processor.

| Process | Allocation | | | Max | | | Available | | | Need | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C | A | B | C |
| $P_0$ | 0 | 1 | 0 | 7 | 5 | 3 | 3 | 3 | 2 | 7 | 4 | 3 |
| $P_1$ | 2 | 0 | 0 | 3 | 2 | 2 | | | | 1 | 2 | 2 |
| $P_2$ | 3 | 0 | 2 | 9 | 0 | 2 | | | | 6 | 0 | 0 |
| $P_3$ | 2 | 1 | 1 | 2 | 2 | 2 | | | | 0 | 1 | 1 |
| $P_4$ | 0 | 0 | 2 | 4 | 3 | 3 | | | | 4 | 3 | 1 |

Q2. Is the system in safe state?
If yes, the what is the safe sequence?
Apply ~~no~~ the safety algorithm on the given system.

Soln

Safe sequence:

* $i=0$, $P_0$, need $= [7,4,3]$, available $= [3,3,2]$
  need > available
  $P_0$ wait. (must)
* $i=1$, $P_1$, need $[1,2,2]$, available $= [3,3,2]$
  need < available
  $P_1$ can be kept in a safe sequence,
  Update work
  Work = Available + allocation
  $= [3,3,2] + [2,0,0]$
  Work $= [5,3,2]$

⊛ i = 2, $P_2$ = need [6, 0, 0] , available [5, 3, 2]
need > available
$P_2$ must wait
⊛. i = 3, $P_3$ , need [0, 1, 1], available [2, 1, 1]
need < available , update work
work = ~~Available~~ + allocation
work =
work = [5, 3, 2] + [2, 1, 1]
work = [7, 4, 3]
⊛. i = 4 , $P_4$ , need [4, 3, 1], available [7, 4, 3]
need < available , update work
work = + allocation
= [7, 4, 3] + [0, 0, 2]
= [7, 4, 5]

⊛ i = 0, $P_1$ , need [7, 4, 3], available [7, 4, 5]
need < available, update
work = + allocation
= [7, 4, 5] + [0, 1, 0]
work = [7, 5, 5]
⊛ i = 2, $P_2$ , need [6, 0, 0] available [7, 5, 5]
need < available, update
work = + allocation
= [7, 5, 5] + [3, 0, 2]
work = [10, 5, 7]

→ Safe sequence = $P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$
Max = [10, 5, 7]

Exercise :
A system with 5 processores use data structures for allocated , maximum and available resou

as shown in the table. These structures monitor current allocat°, max resource needs and available resources to manage process operations efficiently.

| Process | Allocated | | | | Max | | | | Available | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ |
| | | | | | | | | | 2 | 1 | 0 | 0 |
| $P_1$ | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 2 | | | | |
| $P_2$ | 2 | 0 | 0 | 0 | 2 | 1 | 1 | 1 | | | | |
| $P_3$ | 0 | 0 | 3 | 0 | 6 | 6 | 5 | 4 | | | | |
| $P_4$ | 2 | 3 | 5 | 4 | 4 | 3 | 5 | 6 | | | | |
| $P_5$ | 0 | 3 | 3 | 2 | 0 | 6 | 5 | 2 | | | | |

a) Compute the need matrix DS

b) Determine all possible safe sequence.

**Soln:**

a)

need = max − allocat°

| Process | Need | | | |
|---|---|---|---|---|
| | $R_1$ | $R_2$ | $R_3$ | $R_4$ |
| $P_1$ | 0 | 0 | 0 | 0 |
| $P_2$ | 0 | 1 | 1 | 1 |
| $P_3$ | 6 | 6 | 2 | 4 |
| $P_4$ | 2 | 0 | 0 | 2 |
| $P_5$ | 0 | 3 | 2 | 0 |

(b). Safe sequence

- $i = 1$, $P_1$, need [0,0,0,0], available [2,1,0,0]

  need < available,

  $P_1$ executed / processed

  Update available = available + allocation

  = [2,1,0,0] + [0,0,1,2]

  = [2,1,1,2]

- $i = 2$, $P_2$, need [0,1,1,1], available [2,1,1,2] need < available

$P_2$ processed

update available = + allocation

= [2, 1, 1, 2] + [2, 0, 0, 0]

= [4, 1, 1, 2]

⊗. $i = 3$, $P_3$, need [6, 6, 2, 4], available [4, 1, 1, 2]

need > available

$P_3$ must wait (enqueue)

⊗. $i = 4$, $P_4$, need [2, 0, 0, 2], available [4, 2, 1, 2]

need < available

$P_4$ processed / executed

new available = + allocat°

= [4, 2, 1, 2] + [2, 3, 5, 4]

= [6, 5, 6, 6]

⊗ $i = 5$, $P_5$, need [0, 3, 2, 0], available

need < available

$P_5$ executed

update available = + allocation

= [6, 5, 6, 6] + [0, 3, 3, 2]

= [6, 8, 9, 8]

⊗ $i = 3$, $P_3$, need [6, 6, 2, 4], available

need < available

$P_3$ executed / processed

update available = + allocat°

= [6, 8, 9, 8] + [0, 0, 3, 0]

= [6, 8, 12, 8]

→ safe sequence = $P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_3$

$P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_3 \rightarrow P_5$