25.06.22
Saturday

local search

- uninformed & informed search — path is necessary
  - finding the exact goal was necessary
  - ~~fin~~ state space was finite
- but these exclude a large number of problems.
  - ↳ state space infinite
  - ↳ finding path not necessary
  - ↳ don't need exact goal — just get close enough

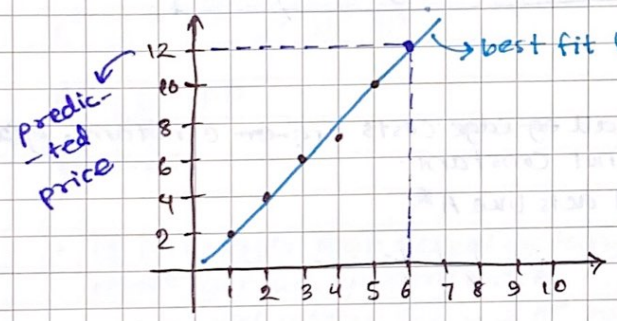
} local search

↳ regression analysis is done w/ local search

x days	y onion price
1	2
2	4
3	6
4	7
5	10
6	?

→ design an agent that will predict the price on 6th day based on past data.

- plot in a graph.



$y = mx \Rightarrow y = 2x$

- y → onion price (given)
- m → unknown
- x → day (given)

the agent needs to figure out this function

↓

search for m

- domain of the search space for m → real numbers $\mathbb{R}$

↓

infinite state space.

Hill climb search:

- take a random value for m, and check correctness.

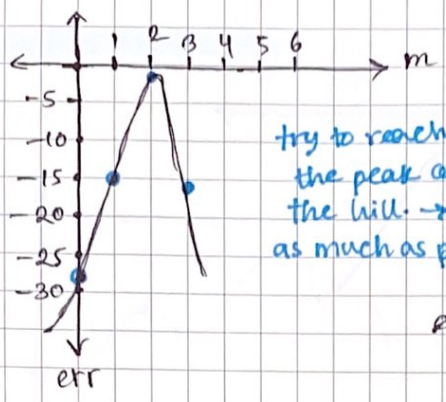
if $m = 1$,
- $x = 1 \rightarrow y = 1$
- $x = 2 \rightarrow y = 2$
- $x = 3 \rightarrow y = 3$
- $x = 4 \rightarrow y = 4$
- $x = 5 \rightarrow y = 5$

} doesn't match w/ actual y value. ∴ find error.

~~error = (|2 − 1| + |4 − 2| + |6 − 3| + |7 − 4| + |10 − 5|)~~

$error = -(|1-2| + |2-4| + |3-6| + |4-7| + |5-10|) = -15$

- then try plotting the error.



try to reach the peak of the hill. → go up as much as possible.

if $m=0$,
$err = -(|0-2|+|0-4|+|0-6|+|0-7|+\dots)$
$= -29$
error increasing, so m going down isn't the right direction.

now, if $m=2$
$err = -(|2-2|+|4-4|+|6-6|+|7-8|+(10-10))$
$= -1$
if $m=3$, $err = -(|2-3|+|4-6|+|6-9|+|7-12|+|10-15|)$
$= -16$
error increasing again
so $m=2$ is goal.

days	price	m=1	m=0	m=2	m=3
1	2	1	0	2	3
2	4	2	0	4	6
3	6	3	0	6	9
4	7	4	0	8	12
5	10	5	0	10	15
6	?				

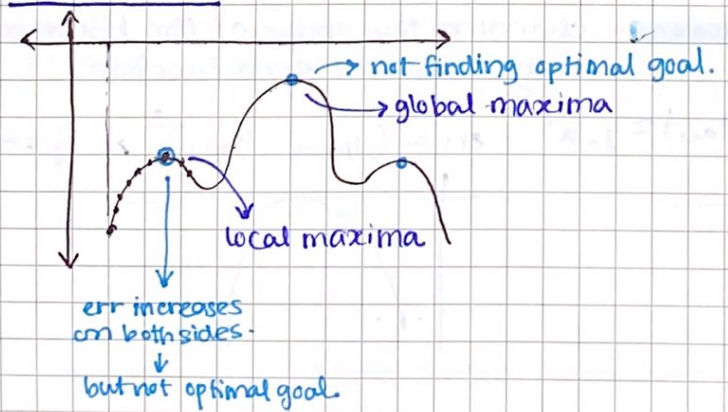
↑ actual value

$$err = \sum_{n=1}^{n} |y_{actual} - y_{pred.}|$$

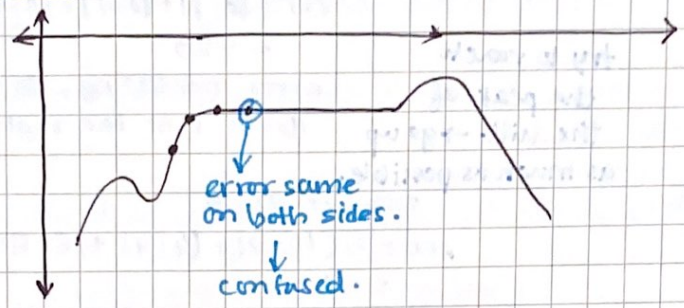
- $m=2$ doesn't get the perfect goal, but close enough
- how we got to $m=2$ doesn't matter → path doesn't matter.
- $m \rightarrow [-\infty, \infty]$ → infinite state space.

- finding the peak of the hill → **hill climb search.** → simplest local search.
  - used (-) in err function to make it a hill.
- only 1 unknown value → 2D hill
  2 unknowns → 3D hill
  → $y = mx + b$ → m and b unknown
- disadvantages / weaknesses:
  1. **local maxima**
  2. **Plateau**
  3. **Ridges**

1. Local Maxima:



2. Plateau:



↳

1	2	5
8	7	4
	6	3

$h = -6$ → local maxima

1	2	3
8		4
7	6	5

→ goal.

h = manhattan distance

up ↓

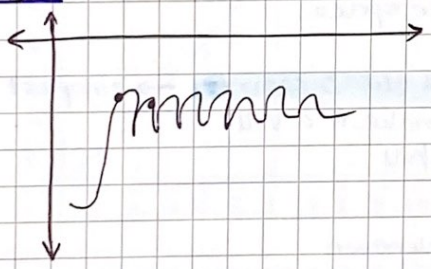
1	2	5
	7	4
8	6	3

$h = -7$

1	2	5
8	7	4
6		3

$h = -7$ → error increasing on both sides.
→ same on both sides → plateau

3. Ridges:



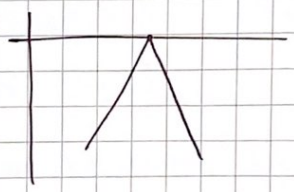
lots of local maxima beside e/o
– confusing.

Solutions:

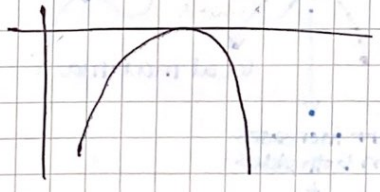
- **Random Restart** – find another random starting point and rerun the search.
  - return the best value of the different searches.
  - increases possibility of getting global maxima, or gives the best local maxima at least.

- **Problem reformulation** – changing the shape of the hill itself.
  - using a different error function

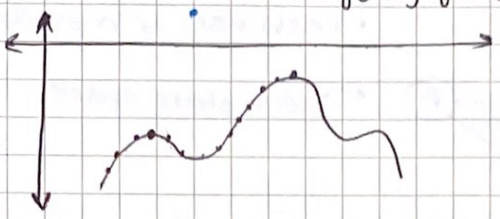
$err = -|y_{pred} - y_{act}| \approx y = x^2$



$err = (y_{pred} - y_{act})^2 \approx y = -|x|$



- **simulated annealing** – uses a temperature value. that ~~increases~~ gets lower throughout the search.
  - lower temp → lower error acceptance.
  - going further after finding the first local maxima.



- temp initially high, gets lower w/ each step, goes until $temp_{min}$ is reached.

check slide!

02.07.22 Saturday

Constraint Satisfaction problem

~~constra~~ The searches we learned so far goes through the entire state space.

- CSP – variables w/ assigned value within a domain that is assigned using a set of constraints to reach a goal.
- reducing the state space based on constraints.

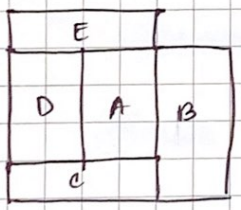
↳ pre advising: generated routine – final goal.

- there are some given constraints – final clash, class clash, pre requisite, credit limit.....etc..

Pre-ad:

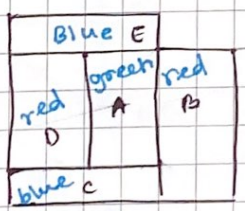
- variable – ~~different time slots~~ weekday time slot
- domain – course sections
- constraints – no theory clash, no final clash, pre-req completion, credit limit.
- goal – final routine

Map coloring problem:



color w/ Red, Green, Blue.
can't have 2 adjacent same colors.

- variable – A, B, C, D, E
- domain – {red, green, blue}
- constraints – adjacent regions can't have same color.
  – $E \neq D$, $E \neq A$, $A \neq B$, $A \neq C$, $A \neq D$, $D \neq C$, $B \neq C$
- goal –

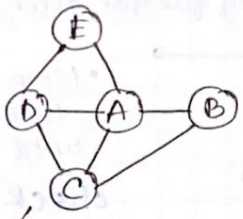


↳ a sol^n solution (can have multiple)

- csp is done through **backtracking**

steps:

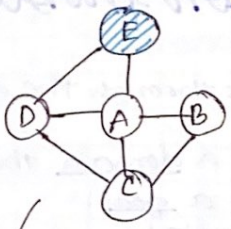
1. represent w/ a **constraint graph**.



- each var - is a node.

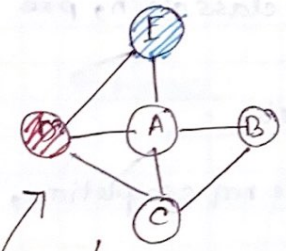
→ root state space.

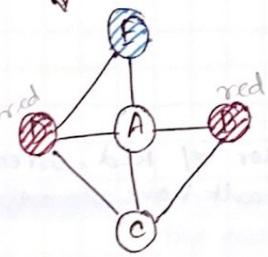
2. apply a simple searching algo ~~on~~ on the empty constraint graph.

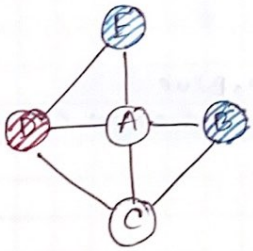


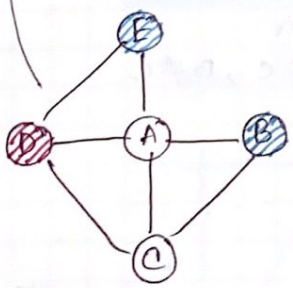
*effective but not efficient.

- randomly picking variables and assigning values.

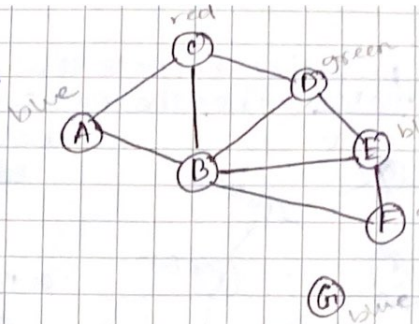




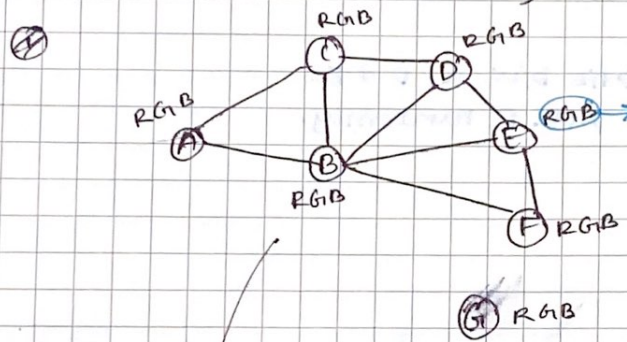




Dead end.
So backtrack.

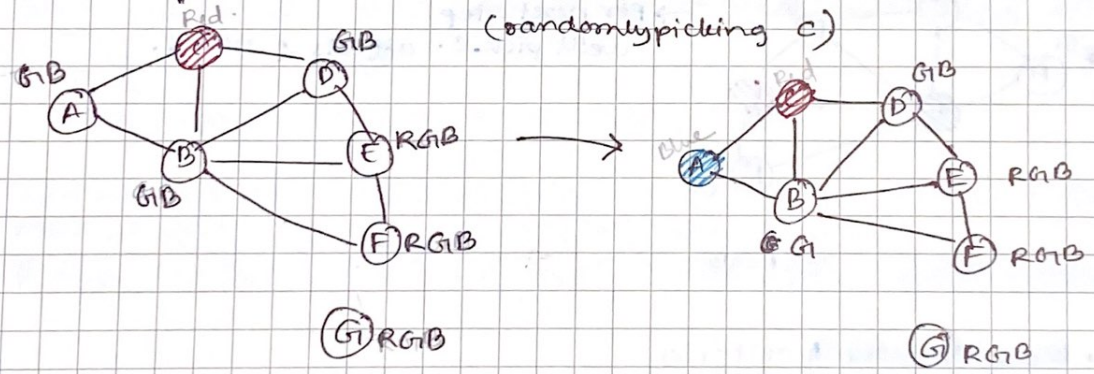


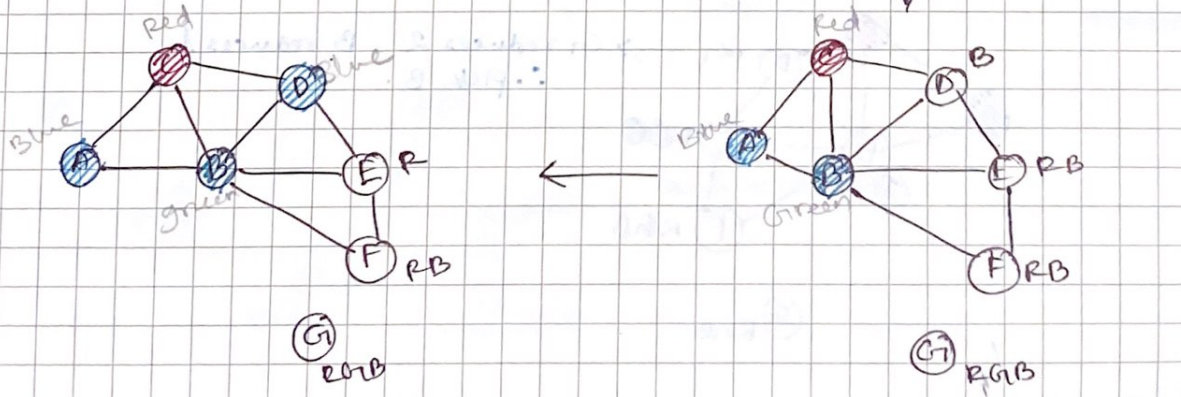
- in order to not choose randomly, need some criteria. This system of choosing w/ criteria – variable & value ordering.
- criteria: 1. Most constrained variable
  2. Most constraining variable
  } variable ordering



Most constrained variable – pick the one w/ the smallest domain.

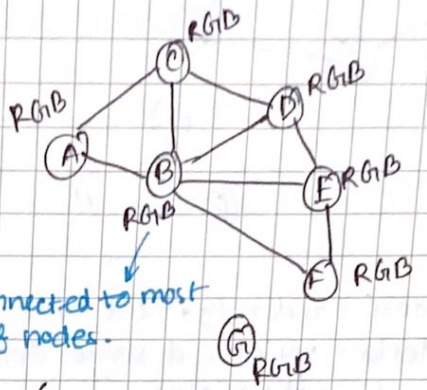
- if equal pick randomly.





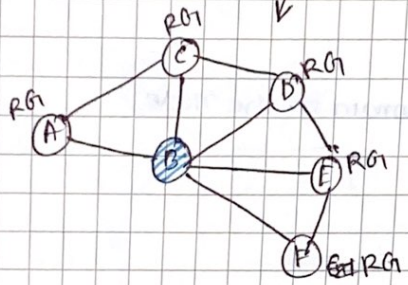
Most constraining variable – prioritize the node that affects the most nodes → has the most connections

*also called **degree heuristic**

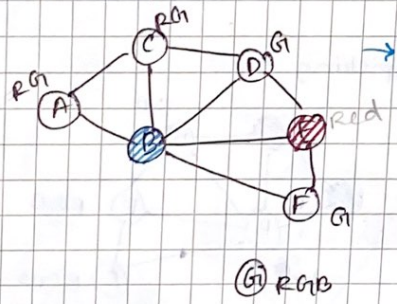


* the most number of UNASSIGNED nodes *

Connected to most # of nodes.

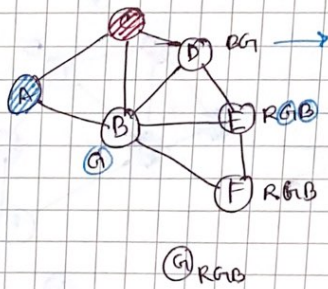


→ tie b/w C, D & E
pick E randomly.



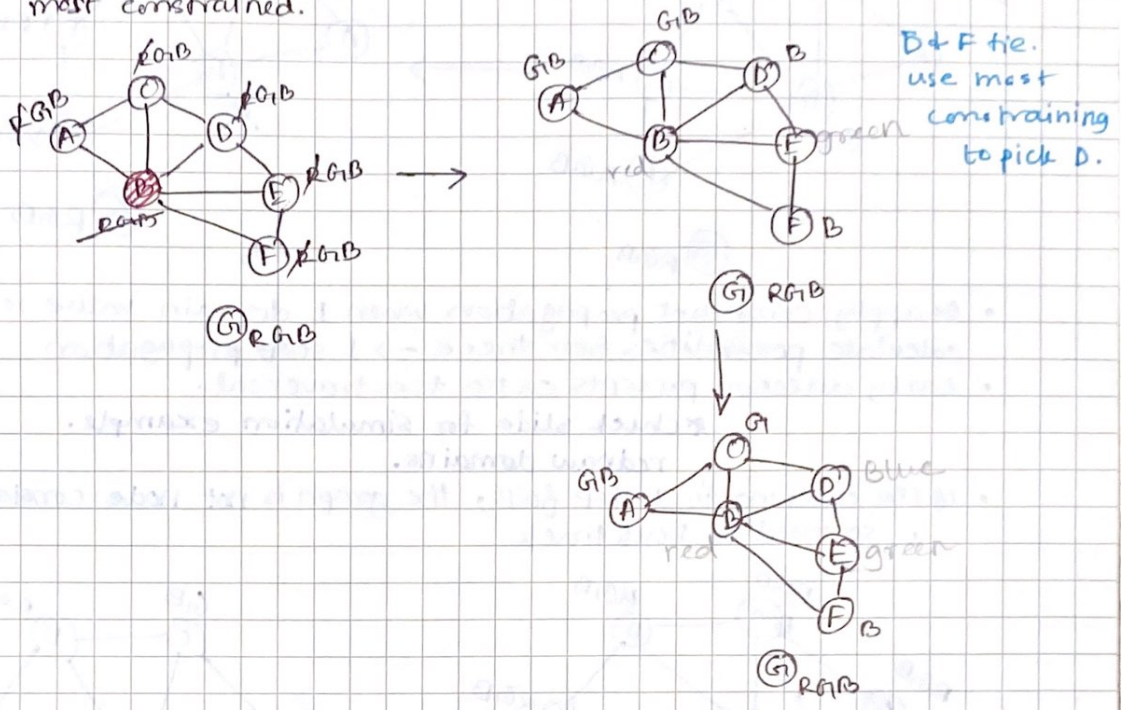
→ For next step we'll pick C. affects 2 nodes.

- **Value ordering** criteria:
  1. **least constraining value** – reduces the least no. of values in other nodes



→ G reduces 2, B reduces 1.
∴ pick B.

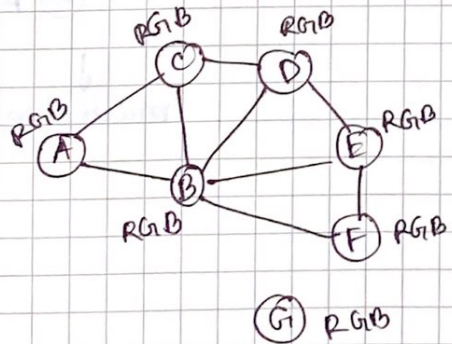
- combo of all 3 used irl.
  - initially pick node w/ most constraining var. to break the tie of most constrained.



16.07.22
Saturday

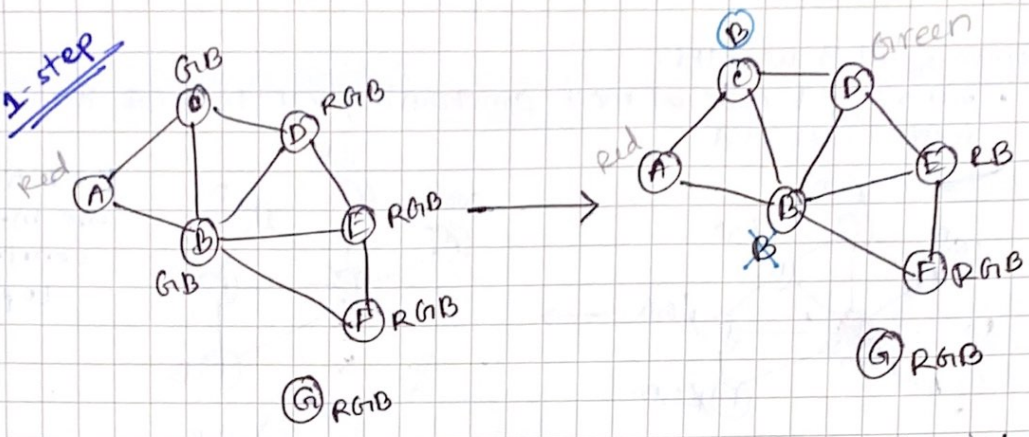
- Most constrained → most constraining → least constraining value
  - most constraining var.
  - most constraining for tie breaker initially

- can apply backtracking in advance if we know we'll hit a dead end.
- more ~~of~~ efficiency if we can predict dead ends early.
- A approach for dead end prediction → constraint propagation
  - 1 step (Node consistency check)
  - Multi step (Arc consistency check)



~~RGB RGB RGB~~

1-step

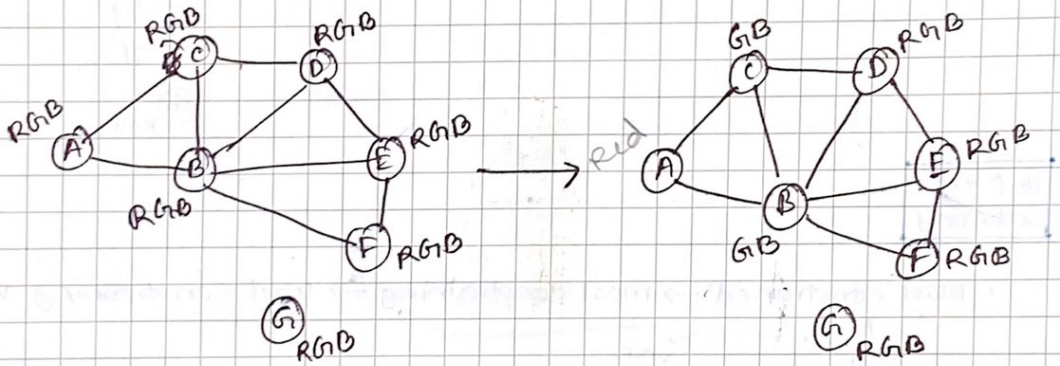


- ~~to~~ apply constraint propagation when 1 domain value is left. calculate possibilities from there → 1 step propagation
- early detection prevents extra tree traversal.

*check slide for simulation example.

redraw domains.

- if the checking in 1 step fails, the graph is not node consistent.
  - so need to backtrack.

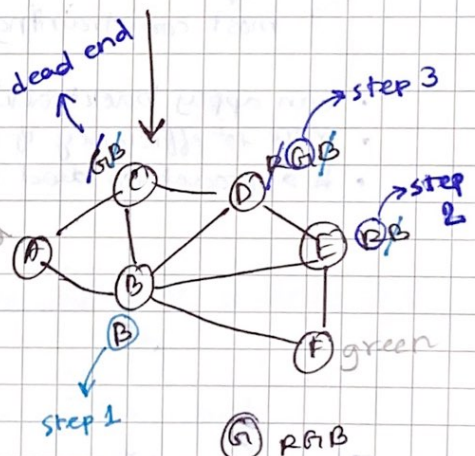


	A	B	C	D	E	F	G
	RGB	RGB	RGB	RGB	RGB	RGB	RGB
	Red	GB	GB	RGB	RGB	RGB	RGB
	~~R~~	B	GB	RGB	RB	~~G~~	RGB
1.	Red	B	~~GB~~	~~RGB~~	~~RB~~	~~g~~	RGB
2.	~~R~~		G	~~RG~~	R	~~g~~	RGB
3.	R		~~G~~	G		~~g~~	

steps ↑

↓ no domain left

↓ dead end.



↓ multi step checking