112001039, shubh pareek

OVERVIEW

My code solves kakuro with **backtracking+arc consistency +node consistency +early failure detection**

I binarized the n-ary constraints by introducing an encapsulating variable,which consisted of a domain corresponding to the particular n-ary constraint .

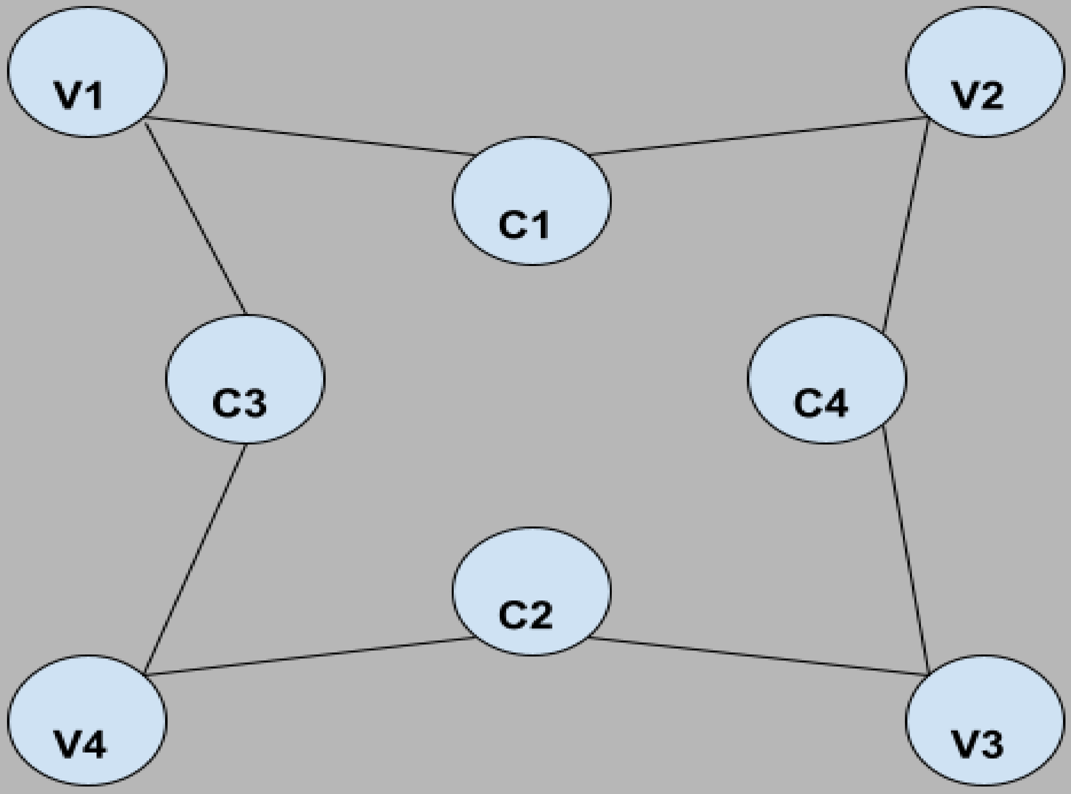
For example take this kakuro instance-

	V-17 ,C3	**V-16 ,C4**
H-16 , C1	**0, V1**	**0, V2**
H-17 , C2	**0, V4**	**0, V3**

Encapsulating variables are- C1{(9,7),(7,9)}, C2{(9,8),(8,9)}, C3{(9,8),(8,9)}, C4{(9,7),(7,9)}

Original variables are - V1,V2,V3,V4

BINARIZED GRAPH



CODE EXPLANATION

FUNCTIONS USED-

Rep function(line -32)- this function will be used to create a dictionary (name-**dic**,line-21,36) that returns a list for domains of variables according to number of variables and sum of the block.
For example -
If sum of block is 17 , and number of variables is 2 then key for dictionary will be "17-2" and list returned by dictionary will be [8, 9]
This function helps in maintaining **node consistency** as the domain for a particular variable is significantly reduced .

Legal assignment function(line-156)- this function implements backtracking by assigning values to the variables in varlist and checking the constraint variables for arc consistency.

Convert function(line-2)-this function converts a string to list , by splitting the string according to commas.
Union function(line -28)- this function unions two lists and returns the union list.
Intersection function(line - 16)-this function intersects two lists and returns the result list.

Code process-

Initially the code reads the input file specified in the s variable(**line** -46), file is opened in the fle variable and its contents are stored as a list of string splitted according to '\n' in reader variable(**line-51**), now from these strings i store values of horizontal and vertical
Constraints in the 2-d list's of name **hsums(line-63)** and **vsums(line-67)** .

Then from line - 118 to line -130 specific horizontal constraints are linked to their variables in consmatrix variable ,the domains and number of variable of a constraint are initialized in a dictionary of name **constraints(line -109,128)**.

From line - 132 to line - 143 the same process as above is repeated but for vertical constraints.

From line-146 to line -147 the domains for variables are reduced by the values stored in **dic** variable calculated by **rep** function.

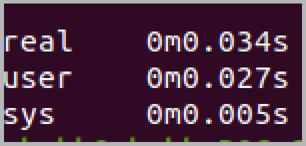
Now the main function **legal assignment** is called on line - 236, which calculates the required kakuro solution.

After that the solution with the question is printed in a file specified on the line-232 .

Observations

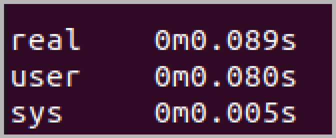
Time taken to solve without node consistency (dic variable optimisation)

Input0.txt - 0.034s, , backtrack count=18



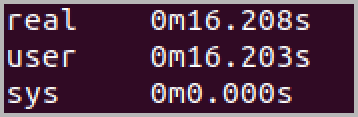
```
real	0m0.034s
user	0m0.027s
sys	0m0.005s
```

Input1.txt - 0.089s, backtrack count= 19717



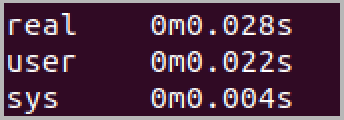
```
real	0m0.089s
user	0m0.080s
sys	0m0.005s
```

Input2.txt - 16.208s, backtrack count=8088390



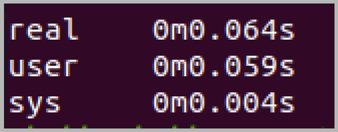
```
real	0m16.208s
user	0m16.203s
sys	0m0.000s
```

Input3.txt - 0.028s, backtrack count=3079



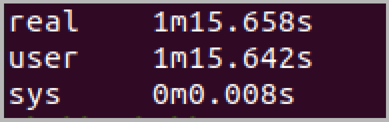
```
real	0m0.028s
user	0m0.022s
sys	0m0.004s
```

Input4.txt - 0.064s, backtrack count=19717



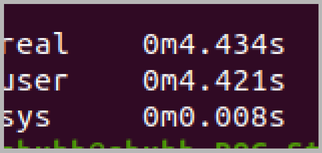
```
real	0m0.064s
user	0m0.059s
sys	0m0.004s
```

Input5.txt - 1m 15.658s, backtrack count=70568812



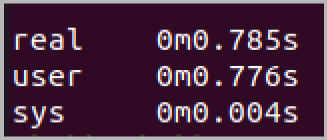
```
real	1m15.658s
user	1m15.642s
sys	0m0.008s
```

Input6.txt 4.434s, backtrack count=3053134



```
real	0m4.434s
user	0m4.421s
sys	0m0.008s
```

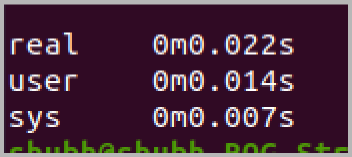
Input7.txt 0.785s, backtrack count=608152



```
real	0m0.785s
user	0m0.776s
sys	0m0.004s
```

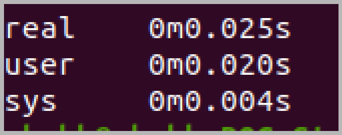
Time taken to solve after adding node consistency(dic variable optimisation)

Input0.txt - 0.022s , backtrack count = 14



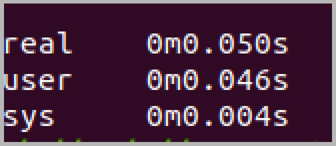
```
real	0m0.022s
user	0m0.014s
sys	0m0.007s
```

input 1.txt -0.025s, backtrack count=996



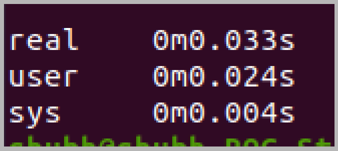
```
real	0m0.025s
user	0m0.020s
sys	0m0.004s
```

Input2.txt - 0.050s, backtrack count=8481



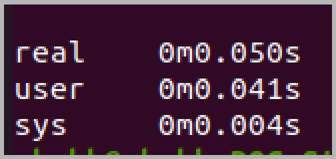
```
real	0m0.050s
user	0m0.046s
sys	0m0.004s
```

Input3.txt - 0.033s, backtrack count=175



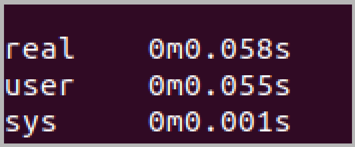
```
real	0m0.033s
user	0m0.024s
sys	0m0.004s
```

Input4.txt - 0.050s, backtrack count=996



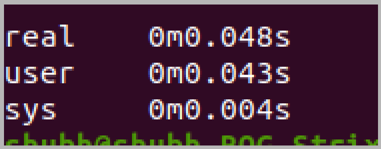
```
real	0m0.050s
user	0m0.041s
sys	0m0.004s
```

Input5.txt - 0.058s, backtrack count=12023



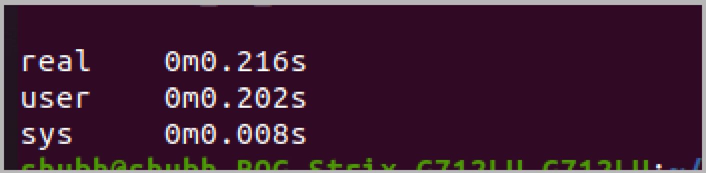
```
real	0m0.058s
user	0m0.055s
sys	0m0.001s
```

Input6.txt - 0.048s, backtrack count=10239



```
real	0m0.048s
user	0m0.043s
sys	0m0.004s
```

Input7.txt - 0.216s, backtrack count=119251



```
real	0m0.216s
user	0m0.202s
sys	0m0.008s
```

From these results it is a obvious conclusion that this optimisation made backtracking significantly faster .