Compiler Construction

BITS Pilani
Pilani Campus

Vinti Agarwal
April 2021

BITS Pilani
Pilani Campus

CS F363, Compiler Construction

Lecture topic: Register Allocation

In previous lectures:

We have seen two kinds of analysis:

- Constant propagation is a *forward analysis*
  - *information is pushed from inputs to outputs*
- Liveness is backward analysis
  - *information is pushed from outputs back towards inputs*

Register Allocation

- Intermediate code uses unlimited number of temporaries
  - simplifies code generation and optimization
  - complicates final translation to assembly
- Typical intermediate code uses too many temporaries

Register Allocation

- The problem

  rewrite intermediate code to use o more

  temporaries than there are machine registers

- The method
  - assign multiple temporaries to each register
  - but without changing the program behavior

Example

$a := c + d$
$e := a + b$
$f := e - 1$

- assume a, f and e are dead after use
  - a dead temporary can be reused
- can allocate a, e and f all to one register (r_1) :

$r_1 := r_2 + r_3$
$r_1 := r_1 + r_4$
$r_1 := r_1 - 1$

- Register allocation is as old as compilers
  - register allocation was used in the original FORTRAN compiler in the 50's
  - very crude algorithm
- A breakthrough came in 1980s
  - register allocation scheme based on graph coloring
  - relatively simple, global and works well in practise

Register Allocation

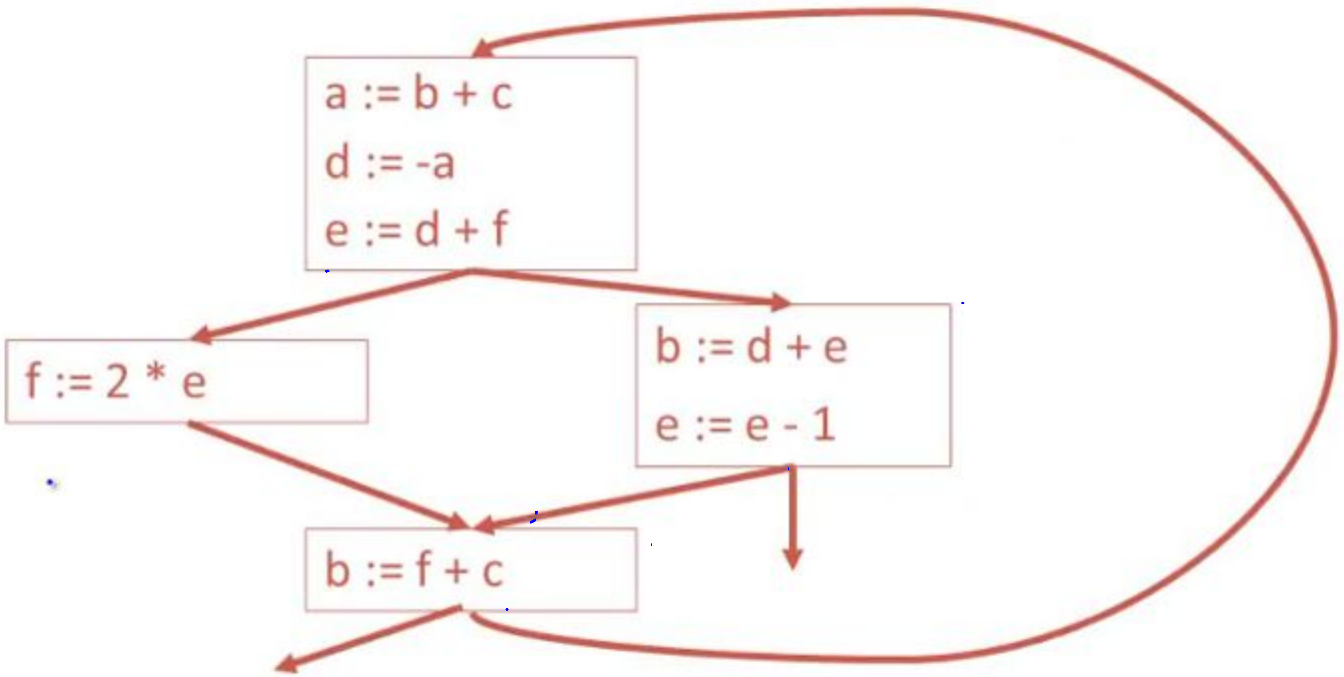
- temporaries t_1 and t_2 can share the same register if at any point in the program at most one of the t_1 or t_2 is live.

or

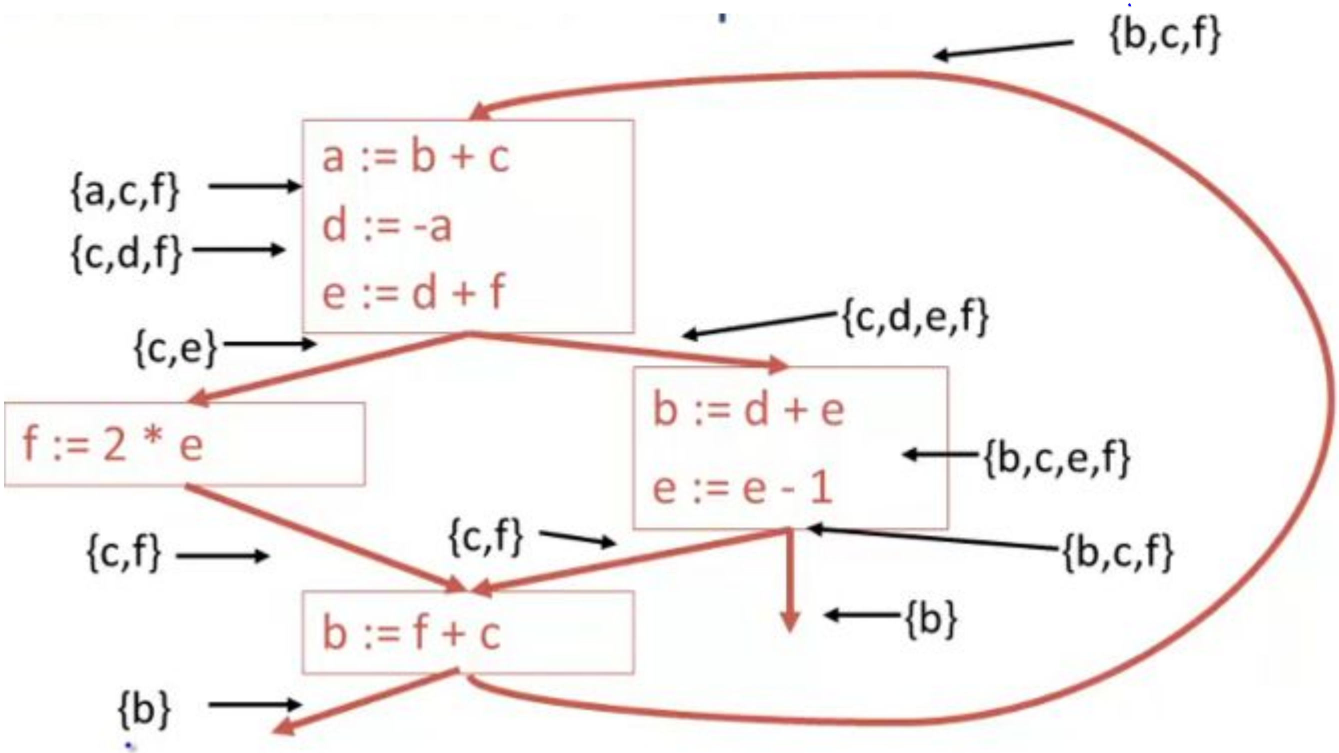
- if t_1 and t_2 are live at the same time, they can not share a register.

Example

- Compute live variables at each point of the program.



Live variables

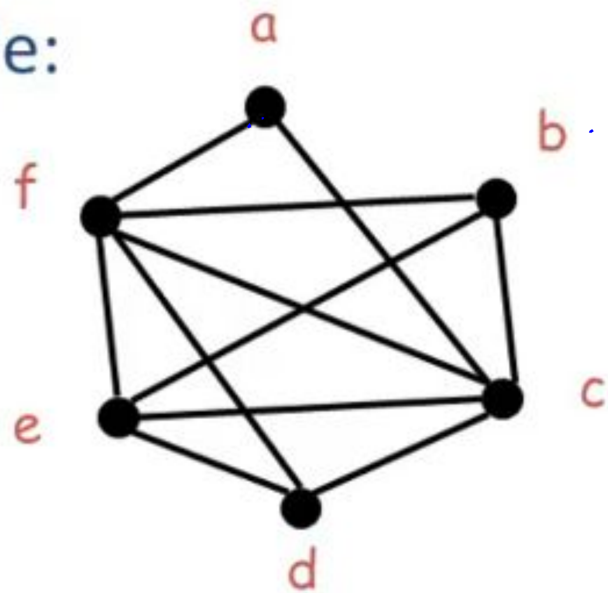


Register Allocation

- Construct an undirected graph
  - a node for each temporary
  - an edge between t1 and t2 if they are live simultaneously at some point in the program
- This is the register interference graph (RIG)
  - Two temporaries can be allocated to the same register if there is no edge connecting them

Register Allocation

- For our example:



- E.g., b and c cannot be in the same register
- E.g., b and d could be in the same register

Register Allocation

- Extracts exactly the information needed to characterize legal register assignments
- Gives a global (i.e. over the entire flow graph (picture of the register requirements.
- After RIG construction the register allocation algorithm is architecture independent

Graph coloring

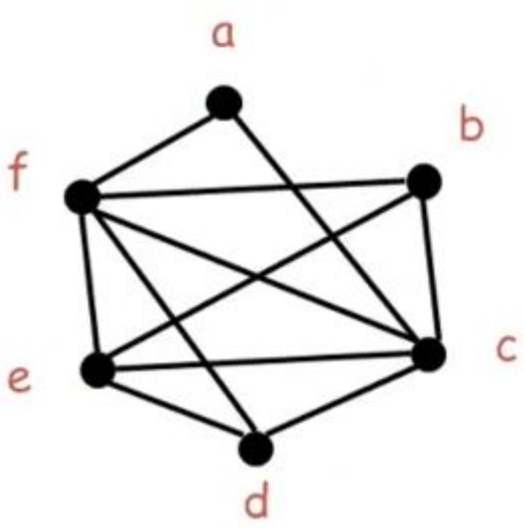
- A coloring of a graph is an assignment of colors to the nodes, such that nodes connected by an edge have different colors
- A graph is k-colorable if it has a coloring with k colors.

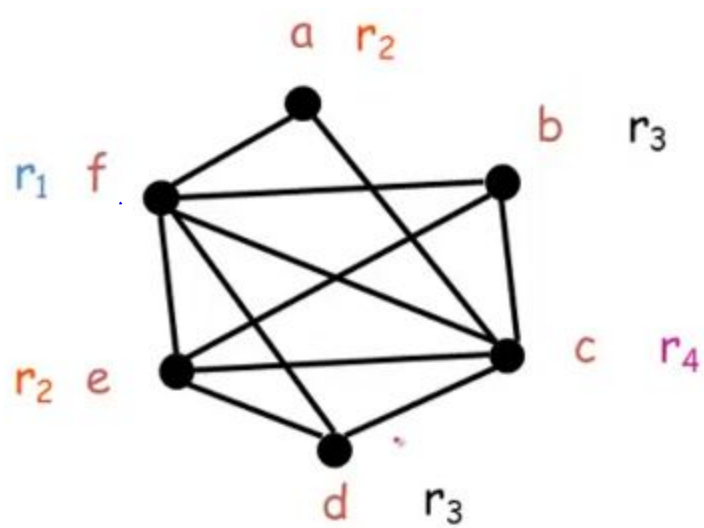
Graph coloring

- In our problem, colors = registers
  - we need to assign colors (registers) to graph nodes (temporaries)
- Let k = number of machine registers
- if the RIG is k-colorable, then there is a register assignment that uses no more than k registers.

Graph coloring

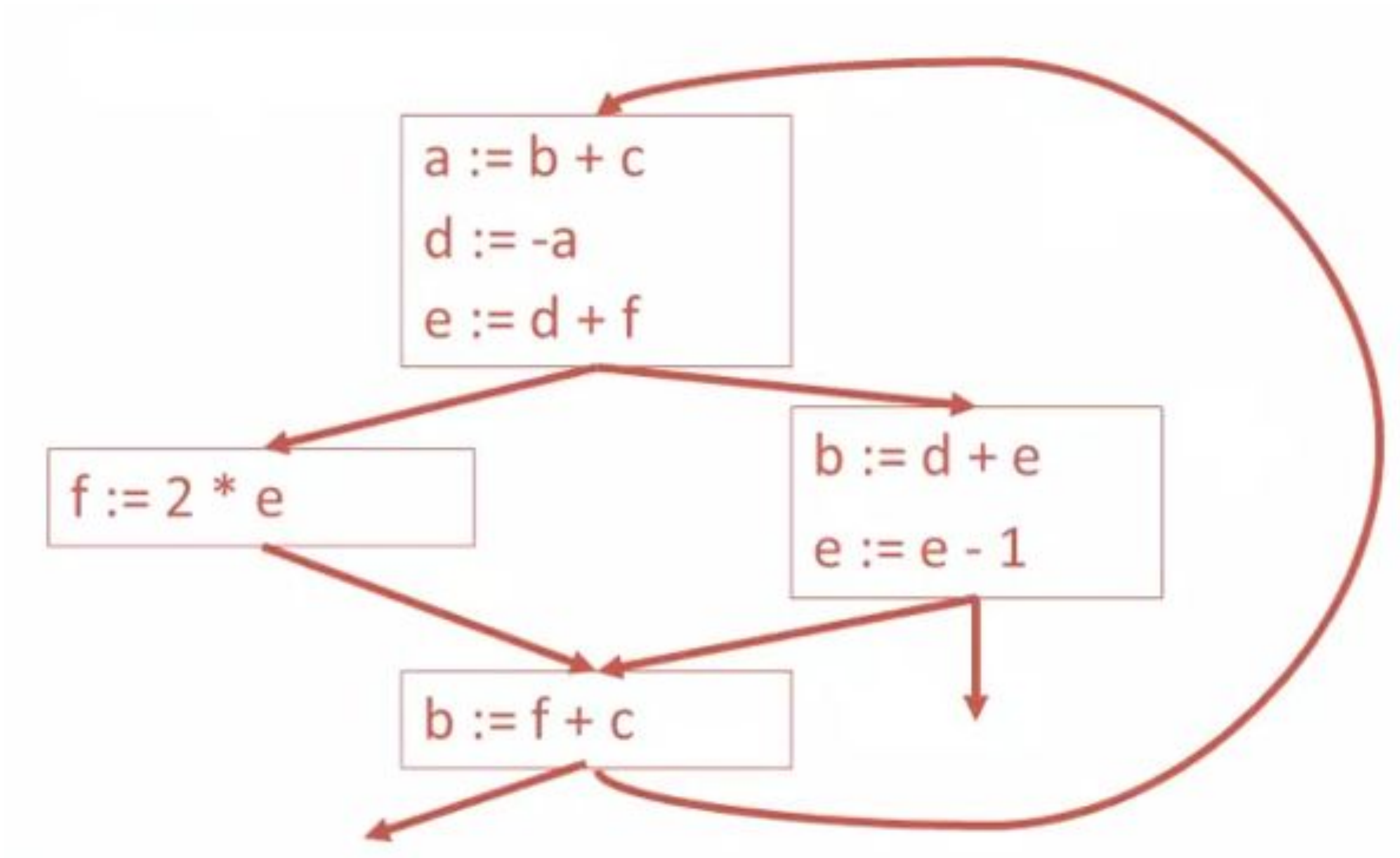
- consider the example RIG



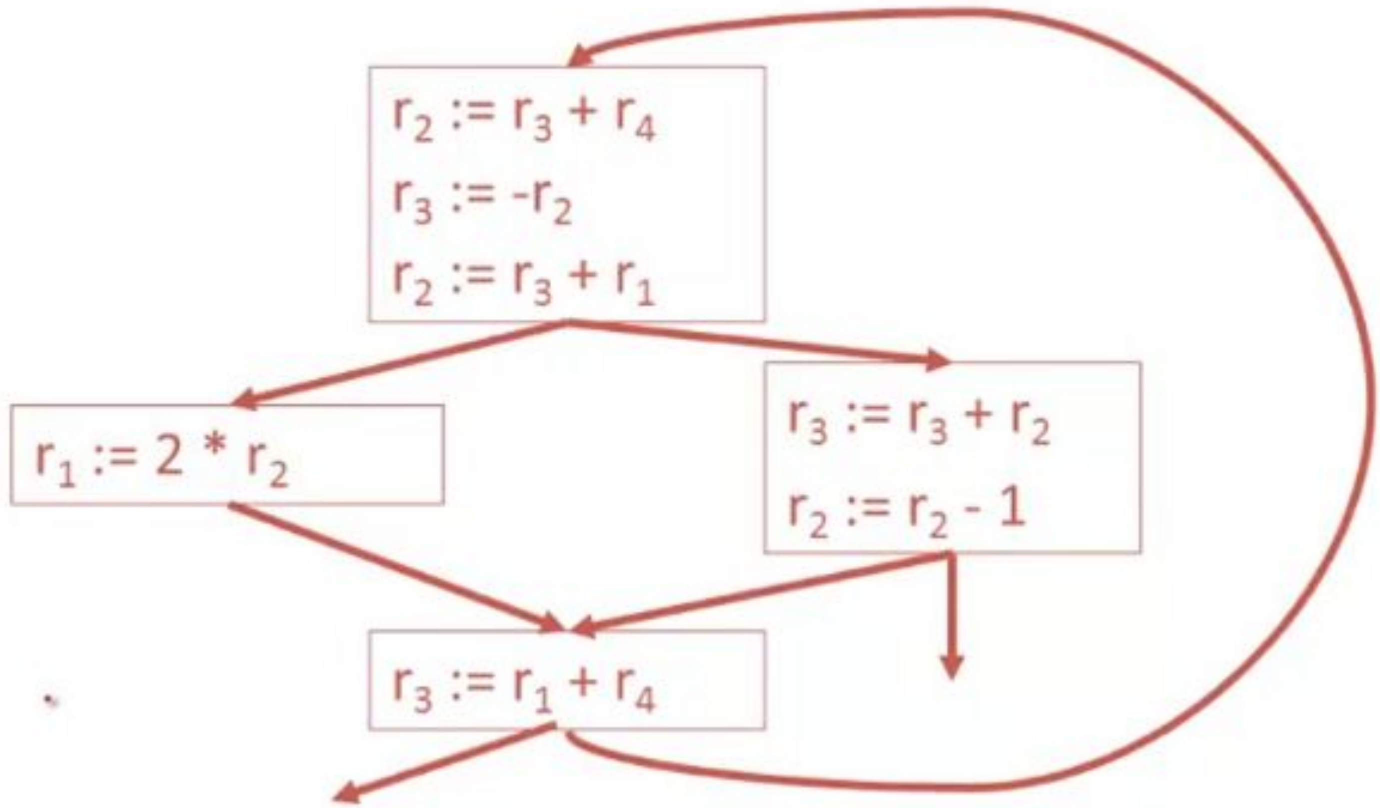


- There is no coloring with less than 4 colors.
- There are 4-colorings of this graph

Graph coloring



Graph coloring



How do we compute graph coloring?

- it isn't easy:
  - This is NP-hard problem. No efficient algorithms are know.
  - One possible solution: use heuristics

- A coloring might not exist for a given number of registers
  - Solution will be discussed later.

Graph coloring Heuristic

- Observation:
  - pick a node t with fewer than k neighbors in RIG
  - eliminate t and its edges from RIG
  - if resulting graph is k-colorable, then so is the original graph

divide and

k = no of registers

- Why?
  - let c_1, c_2,...c_n be the colors assigned to the neighbors of t in the reduced graph

k - n

  - since n < k we can pick some color for t that is different from those of its neighbors

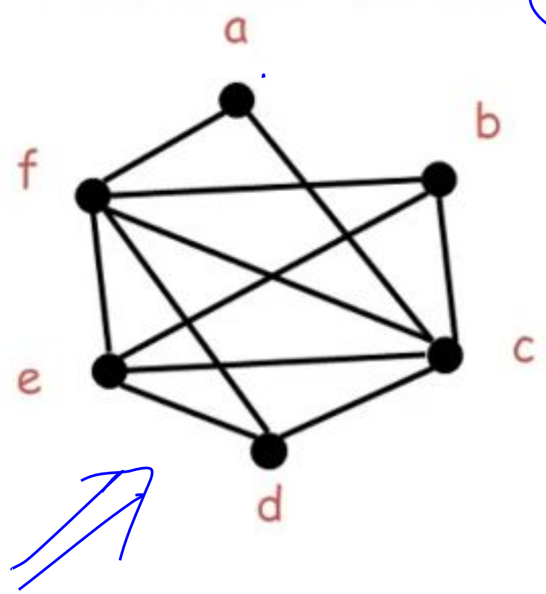
Graph coloring Procedure

- The following works well in practice:
  - pick a node t with fewer than k neighbors
  - put t on the stack and remove it from RIG
  - repeat until the graph is empty
- Assign colors to nodes on the stack
  - start with the last node added
  - at each step pick a color different from those assigned to already colored neighbors

I

Graph coloring example

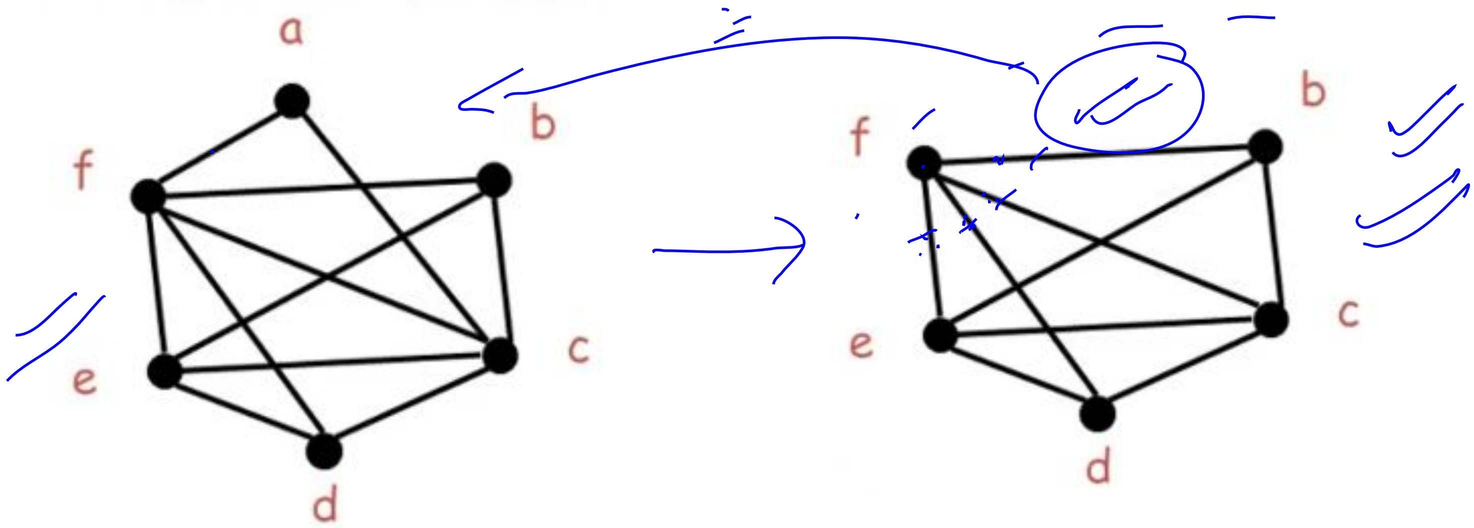
Start with the RIG and with k = 4:



Stack = { a }

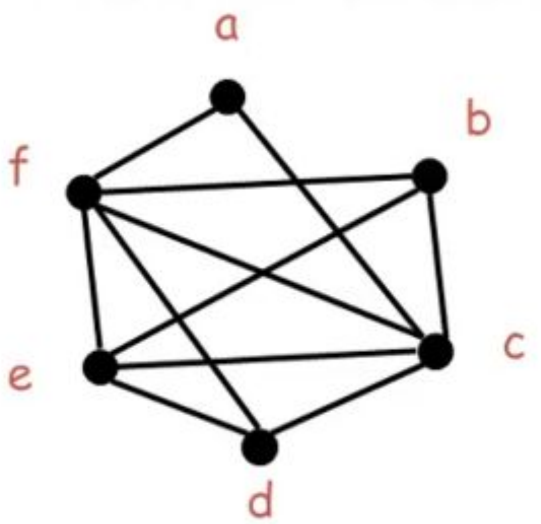
Graph coloring example

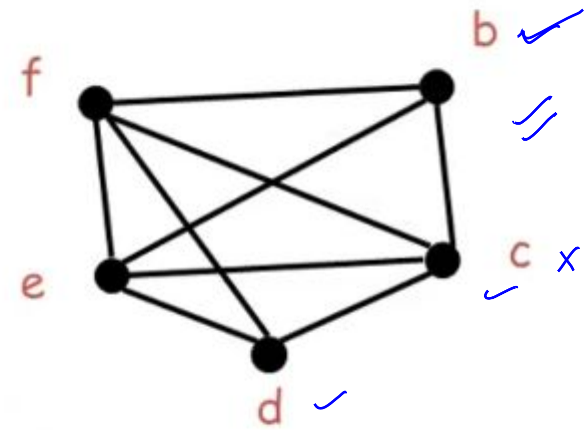
Start with the RIG and with k = 4:

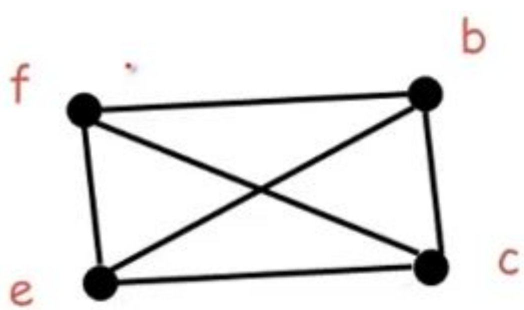


Graph coloring example

Start with the RIG and with k = 4:



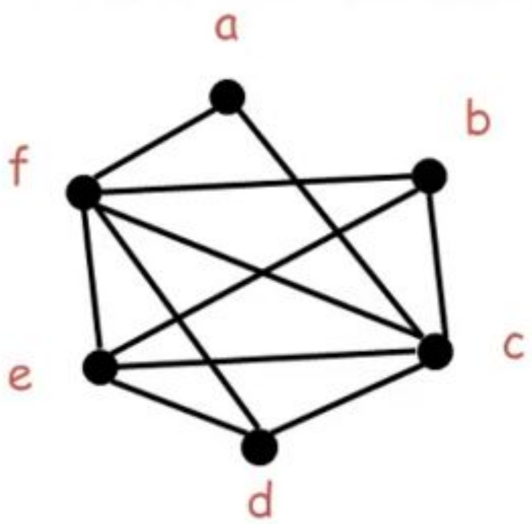


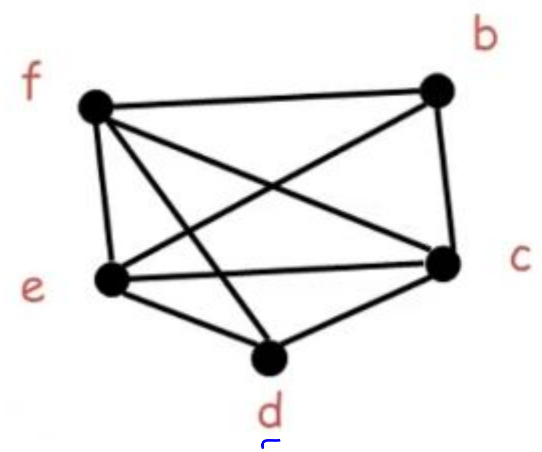


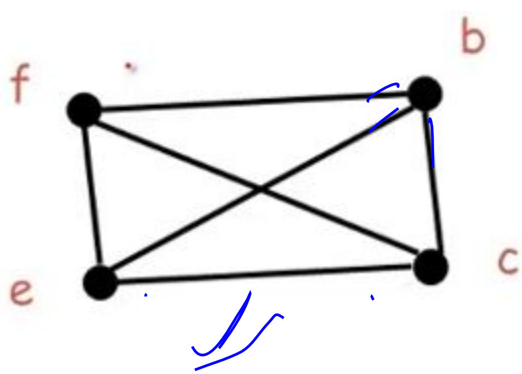
Stack : {d, a}

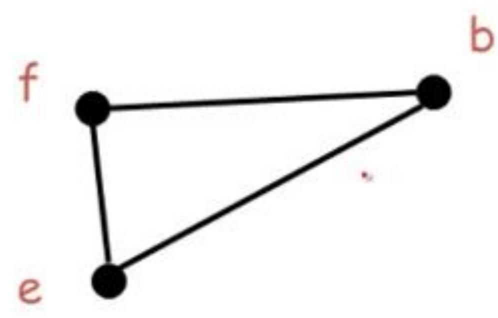
Graph coloring example

Start with the RIG and with k = 4:



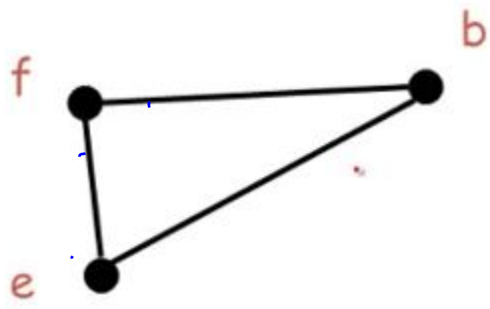






(c, d, a)

Graph coloring example



(b, c, d, a)

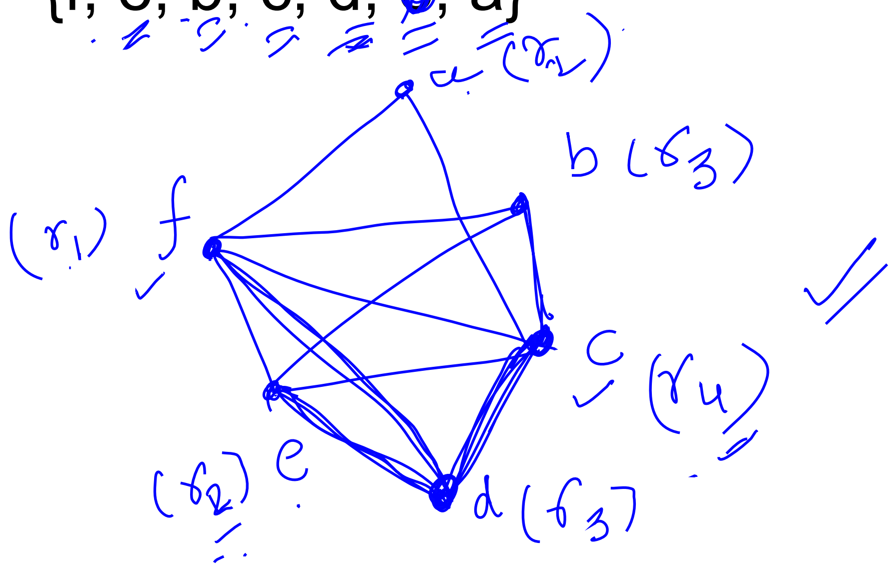
(f, e, b, c, d, a) $\Rightarrow$ stack.

Empty graph- done with the first part !

Graph coloring example

Now start assigning colors to the nodes, starting with the top of the stack

stack = {f, e, b, c, d, e, a}



Thank You!