Conditional Graph

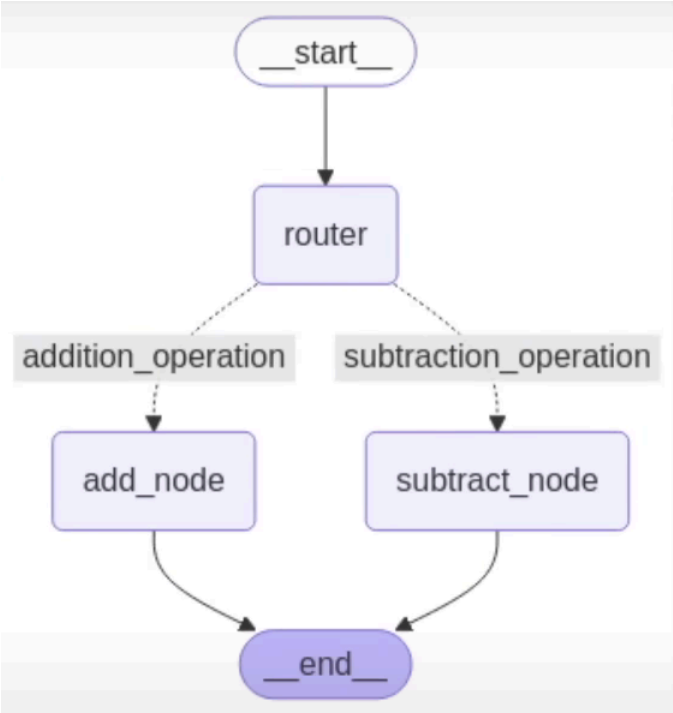
Use → `add_conditional_edges()`

- Use `START` & `END` to manage entry and exit

```
from typing import TypedDict, Dict
from langgraph.graph import StateGraph, START, END
```

```
class AgentState(TypedDict):
    number1: int
    operation: str
    number2: int
    finalnumber: int
```

Workflow:



Create a router:

```
def adder(state: AgentState) → AgentState:
    """This node adds the 2 numbers"""

    state["finalnumber"] = state['number1'] + state["number2"]
    return state

def subtractor(state:AgentState) → AgentState:
   """This node subtracts the 2 numbers"""
   state["finalnumber"] = state["number1"] - state["number2"]
   return state

def decide_next_node(state: AgentState)→ AgentState:
   """This node will select the next node of the graph"""

   if state["operation"]== "+":
      return "addition_operation"

   elif state["operation"] == "-":
    return "subtraction_operation"
```

- `decide_next_node` is a router
- It decides next operation (+ or -)
- **WE CANNOT CALL THE `adder()` OR `subtractor()` FUNCTION INSIDE THIS**
- We have to return the edge

💡 **`addition_operation` and `subtraction_operation` are the names of the adder and subtractor edges resp. They will be defined later in `add_conditional_edges`**

💡 **We are returning the edges and ot state**

```
graph = StateGraph(AgentState)

#NODES
graph.add_node("add_node", adder)
graph.add_node("subtract_node", subtractor)
graph.add_node("router", lambda state:state) #passthrough function

#EDGE
graph.add_edge(START, "router")

#CONDITIONAL EDGES
graph.add_conditional_edges(
   "router",
   decide_next_node,
   {
      # Edge: Node
      "addition_operation": "add_node",
      "subtraction_operation": "subtract_node"
   }
)

graph.add_edge("add_node", END)
graph.add_edge("subtract_node", END)

app=graph.compile()
```

- `graph.add_node("router", decide_next_node)` → **This will not work as we are not returning the state.**

- Therefore, we use `lambda state:state` → **A passthrough function will not change the state**

`graph.add_conditional_edges()`

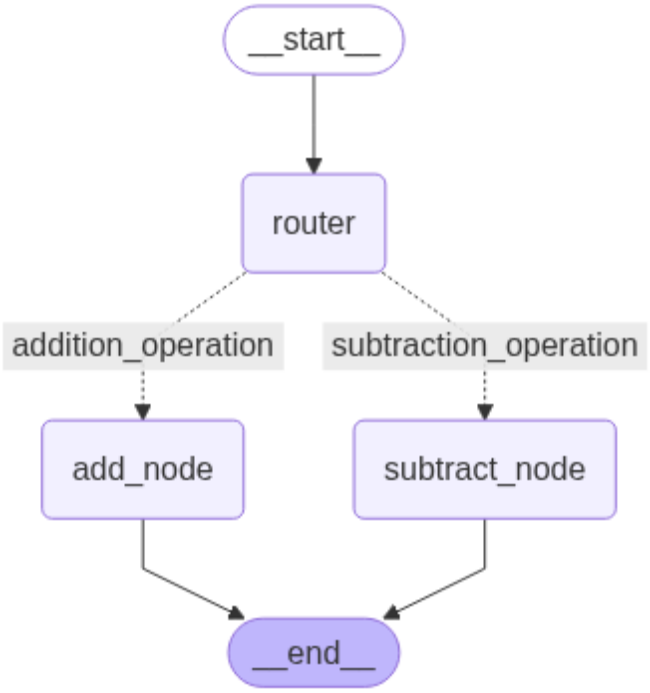
- Provide **Source(Name of the node), Path (Action)** & **Path map**
- `"router"` → Name of the router node
- `decide_next_node` → What it does? (Router Function)
- **Path map: Dictionary**
  - Edges that we earlier passed into router node

> We need 2 edges connecting to END node:

```
graph.add_edge("add_node", END)
graph.add_edge("subtract_node", END)
```

Visualize:

```
from IPython.display import Image, display

display(Image(app.get_graph().draw_mermaid_png()))
```



Invoke:

```
state1 = AgentState(number1=10, number2=2, operation="-")

app.invoke(state1)
```

```
{'number1': 10, 'operation': '-', 'number2': 2, 'finalnumber': 8}
```

OR

```
initial_state_1 = {'number1': 10, 'operation': '-', 'number2': 2}
app.invoke(initial_state_1)
```

```
{'number1': 10, 'operation': '-', 'number2': 2, 'finalnumber': 8}
```

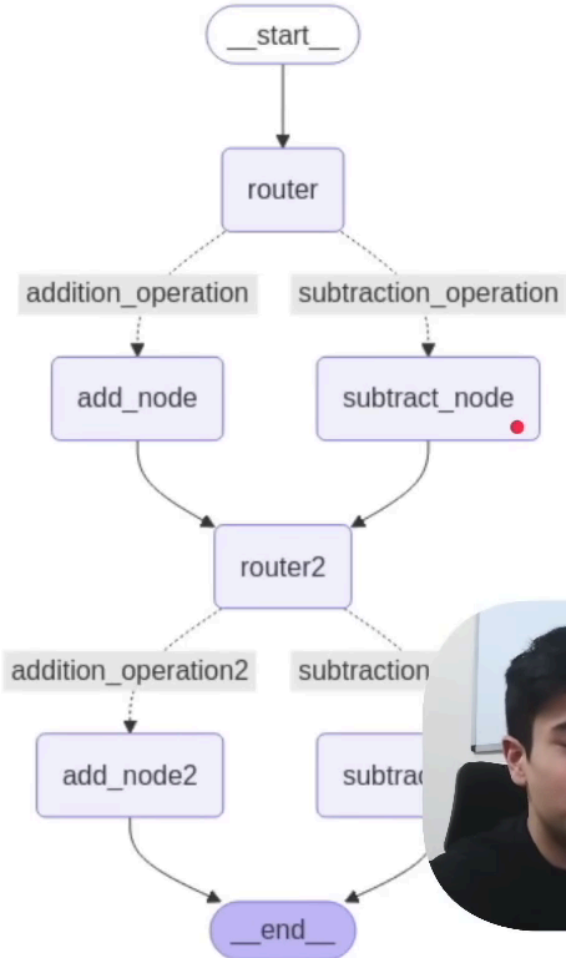
- Same result

Exercise:

Your task:

Make the graph on the right! You will need to make use of 2 conditional edges!

Input: initial_state = AgentState(number1 = 10, operation="-", number2 = 5, number3 = 7, number4=2, operation2="+", finalNumber= 0, finalNumber2 = 0)



- Output: Final result twice

```
from typing import TypedDict, Dict
from langgraph.graph import StateGraph, START, END
```

```
class AgentState(TypedDict):
    number1: int
    number2: int
    number3: int
    number4: int
    operation1 : str
    operation2: str
    result1: int
    result2: int
```

- We defined everything twice

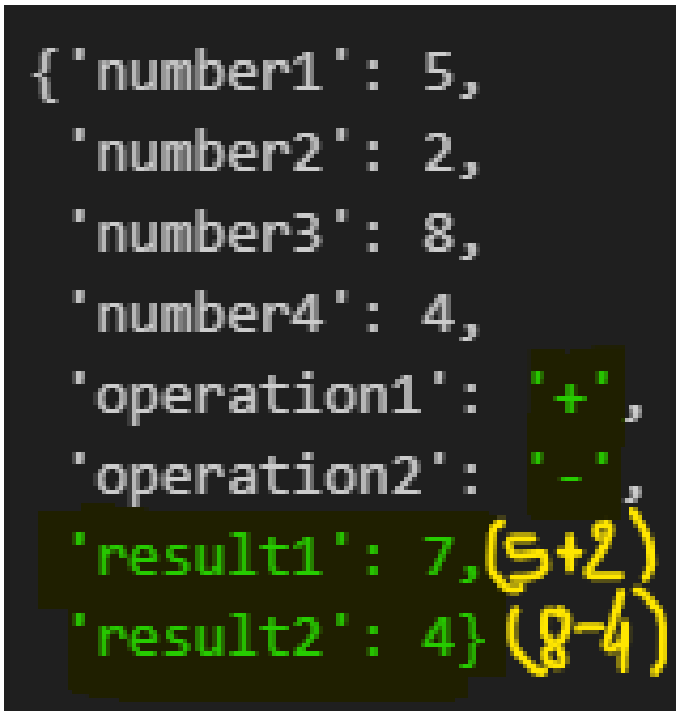
```
graph= StateGraph(AgentState)

graph.add_node("router1", lambda state: state)
graph.add_node("router2", lambda state: state)
graph.add_node("adder1_node", adder1)
graph.add_node("adder2_node", adder2)
graph.add_node("sub1_node", sub1)
graph.add_node("sub2_node", sub2)

graph.add_conditional_edges(
    "router1",
    decide_next_node1,

    {
```

```
    "add_edge1": "adder1_node",
    "sub_edge1": "sub1_node"
    }
   )

graph.add_conditional_edges(
   "router2",
   decide_next_node2,
   {
      "add_edge2": "adder2_node",
      "sub_edge2": "sub2_node"
   }
)

graph.add_edge(START, "router1")
graph.add_edge("adder1_node", "router2")
graph.add_edge("sub1_node", "router2")

graph.add_edge("adder2_node", END)
graph.add_edge("sub2_node", END)


app=graph.compile()
```

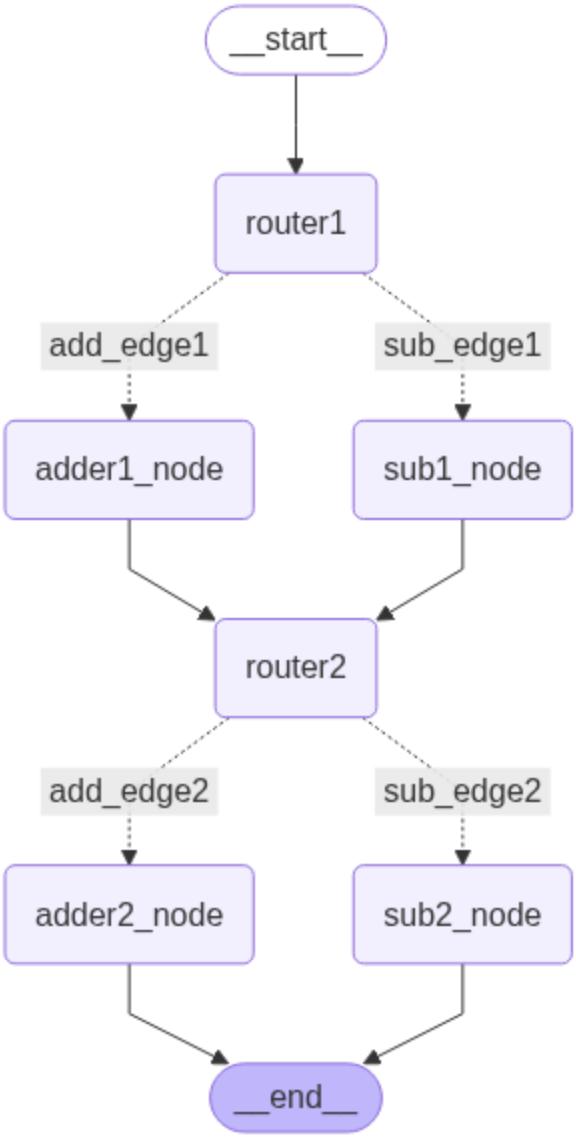
```
state = {
   'number1': 5,
   'number2': 2,
   'number3': 8,
   'number4': 4,
   'operation1' : "+",
   'operation2': "-"
}
```

Invoke:

```
app.invoke(state)
```




```
from IPython.display import Image, display

display(Image(app.get_graph().draw_mermaid_png()))
```


__start__
router1
add_edge1
sub_edge1
adder1_node
sub1_node
router2
add_edge2
sub_edge2
adder2_node
sub2_node
__end__