# Creating a Data Flow Graph

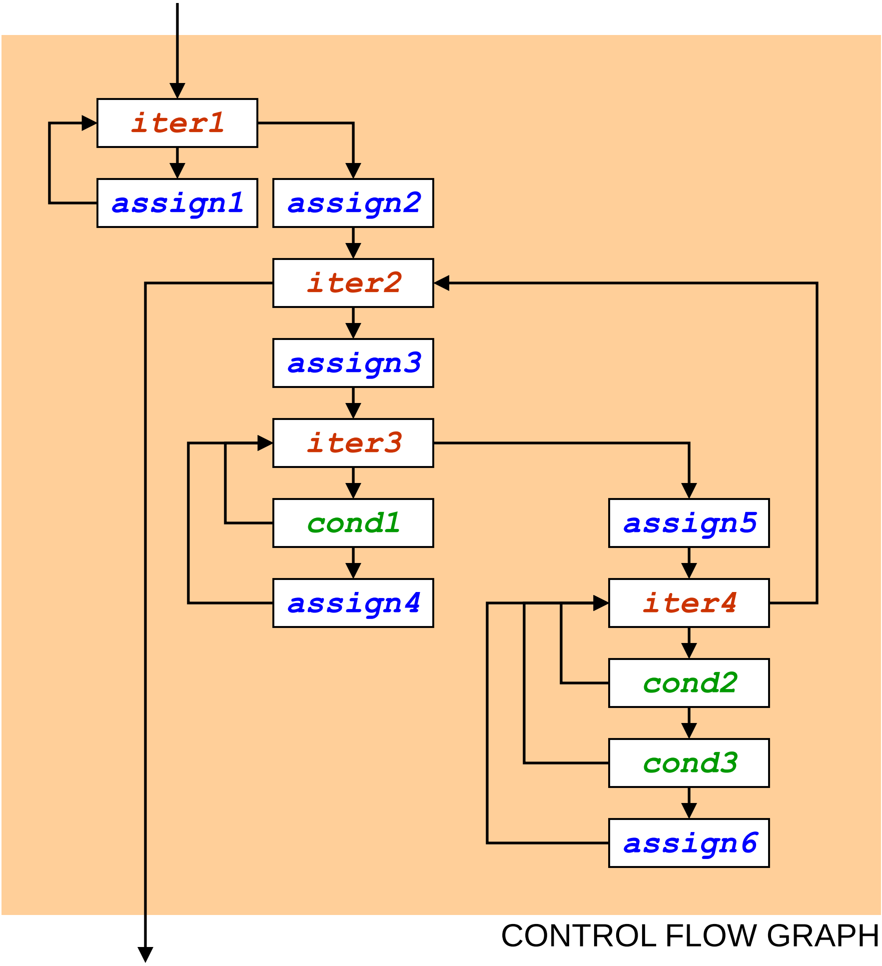
```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (iter1) {
    assign1
  }

  assign2

  for (iter2) {
    assign3
    for (iter3)
      if (cond1)
        assign4

    assign5
    for (iter4)
      if (cond2)
        if (cond3)
          assign6
  }
}
```




CONTROL FLOW GRAPH

# Creating a Data Flow Graph

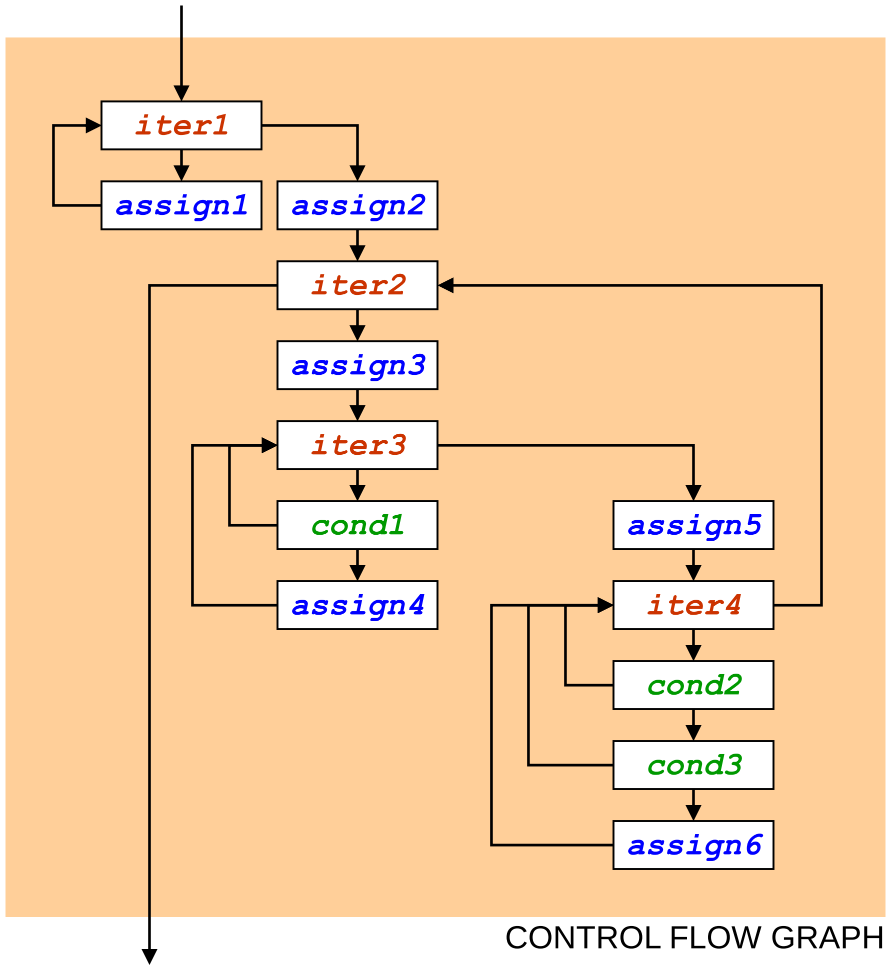
```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (iter1) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (iter2) {
    mini = -1;
    for (iter3)
      if (cond1)
        mini = i;

    visited[mini] = 1;
    for (iter4)
      if (cond2)
        if (cond3)
          d[i] = d[mini] +
                 dist[mini][i];
  }
}
```




CONTROL FLOW GRAPH

# Data Flow Adequacy Criteria

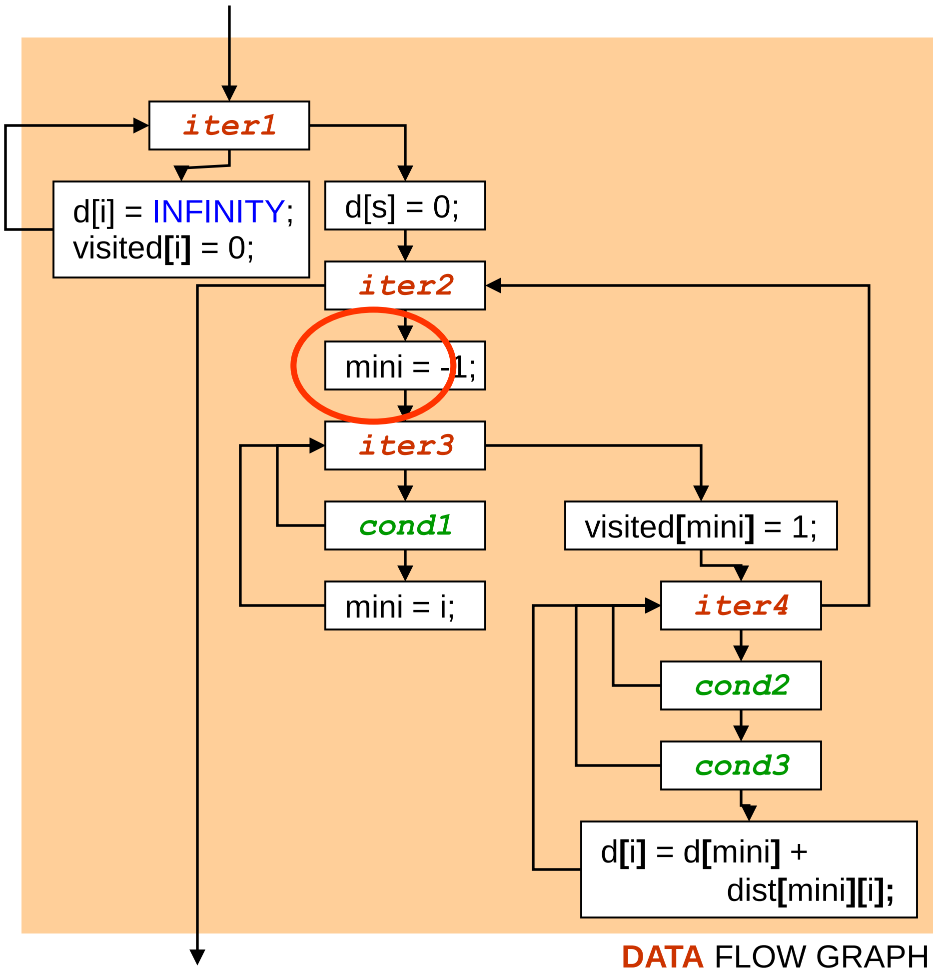
```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (iter1) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (iter2) {
    mini = -1;
    for (iter3)
      if (cond1)
        mini = i;

    visited[mini] = 1;
    for (iter4)
      if (cond2)
        if (cond3)
          d[i] = d[mini] +
                dist[mini][i];
  }
}
```




DATA FLOW GRAPH

# Data Flow Adequacy Criteria

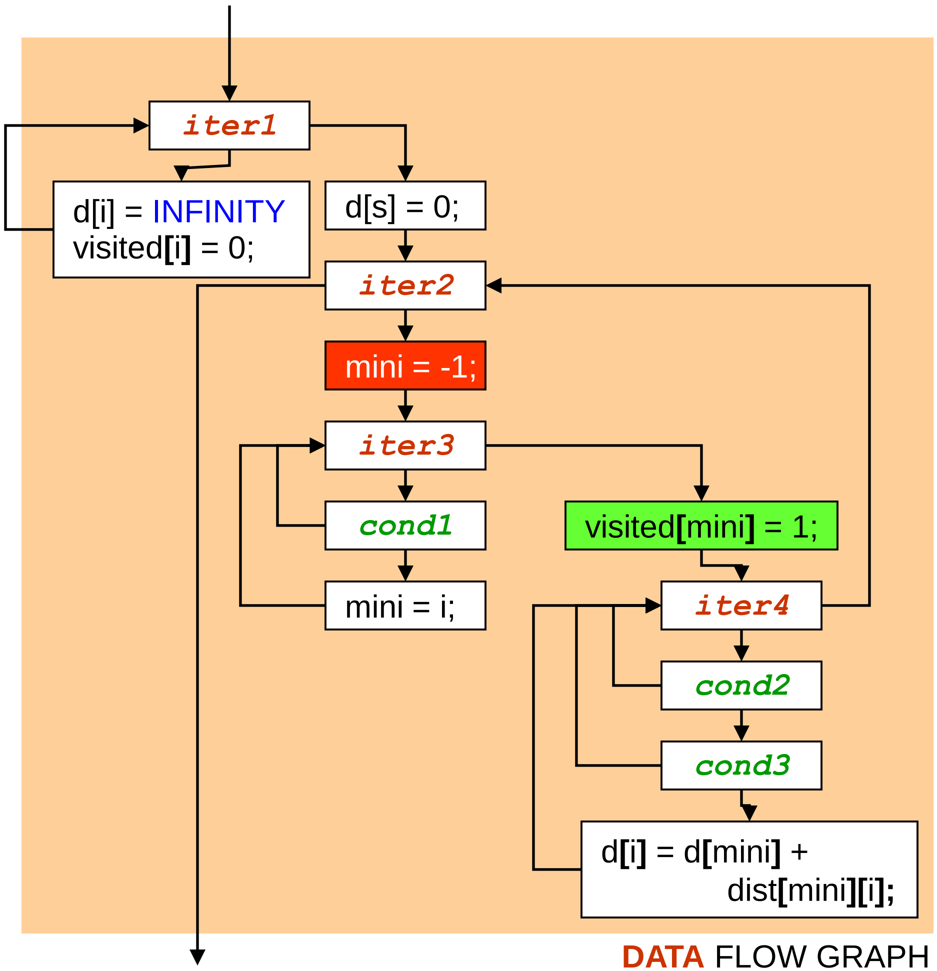
```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (iter1) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (iter2) {
    mini = -1;
    for (iter3)
      if (cond1)
        mini = i;

    visited[mini] = 1;
    for (iter4)
      if (cond2)
        if (cond3)
          d[i] = d[mini] +
              dist[mini][i];
  }
}
```




**DATA** FLOW GRAPH

# Data Flow Adequacy Criteria

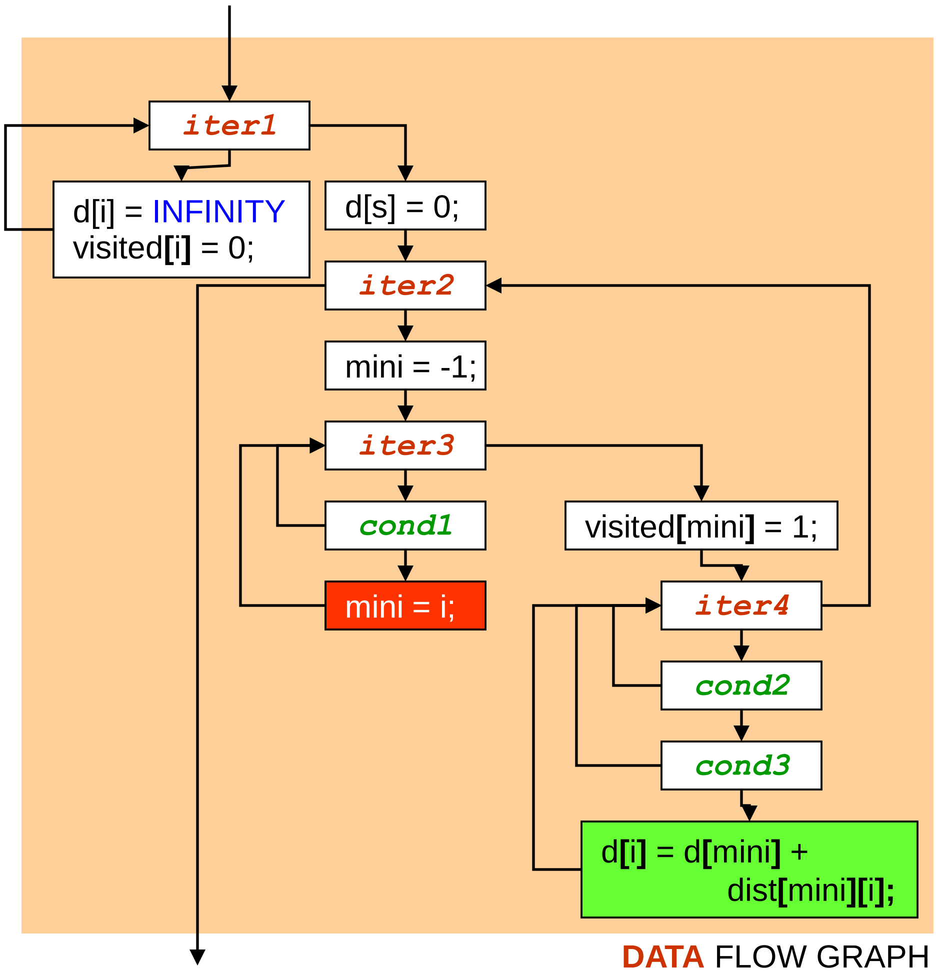
```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (iter1) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (iter2) {
    mini = -1;
    for (iter3)
      if (cond1)
        mini = i;

    visited[mini] = 1;
    for (iter4)
      if (cond2)
        if (cond3)
          d[i] = d[mini] +
              dist[mini][i];
  }
}
```



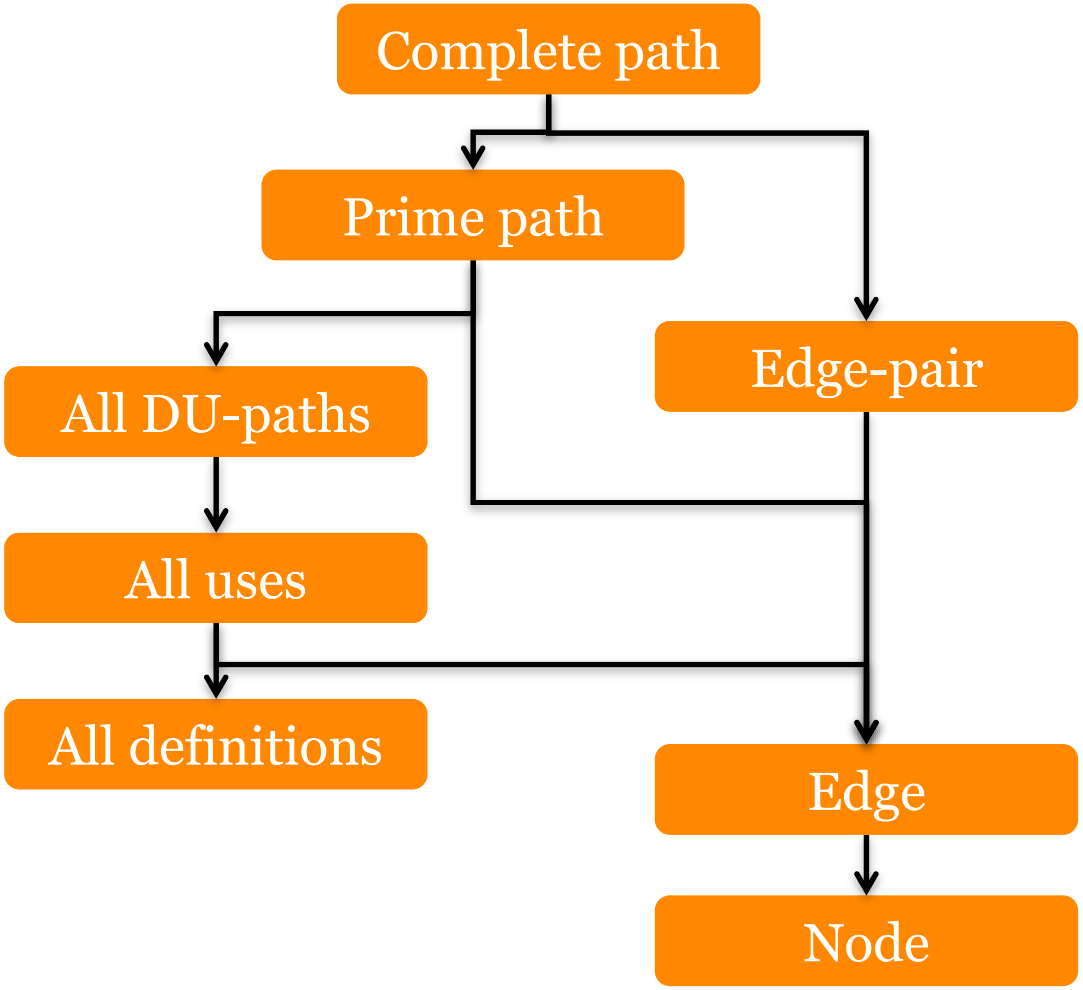

DATA FLOW GRAPH

# Graph-Based Criteria: Subsumption Hierarchy

- **Subsumption hierarchies** theoretically orders adequacy criteria with respect to each other
- Basically, an adequacy criterion A subsumes another adequacy criterion B **if**
  - for all programs P and test suites T,
    - if t in T satisfies A for program p in P
    - then t also satisfies B for p.

# Graph-Based Criteria: Subsumption Hierarchy

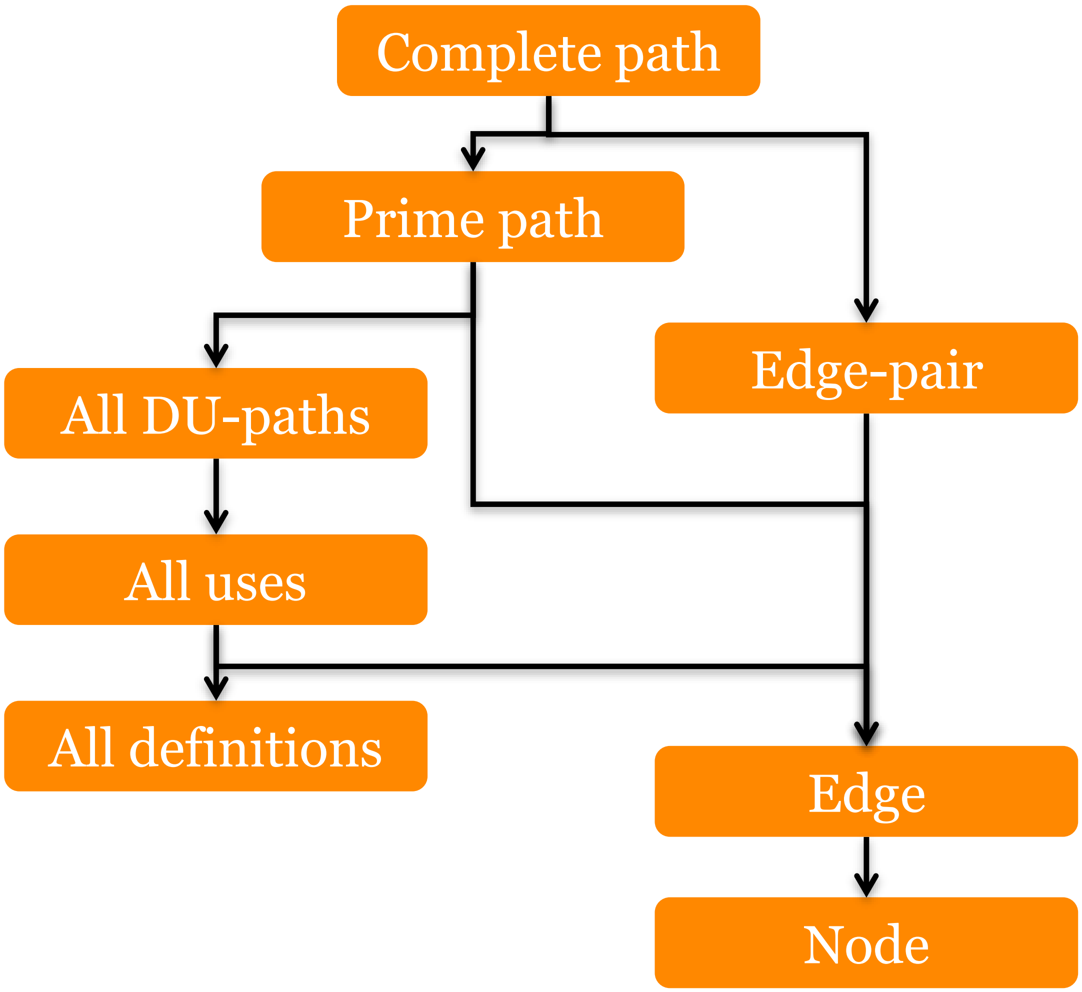
- **A -> B** indicates that **A** subsumes **B**



# Graph-Based Criteria: Subsumption Hierarchy

- **A -> B** indicates that **A** subsumes **B**
- **Question 1:** For a program **P**, two criteria **A** and **B**, and two test suites **t1** and **t2**, if
  - criteria **A** subsumes criteria **B**,
  - test suite **t1** satisfies **A** and
  - test suite **t2** satisfies **B**,

does this mean that **t1** will always reveal at least as many faults as **t2**?



# Graph-Based Criteria: Subsumption Hierarchy

- **A -> B** indicates that **A** subsumes **B**
- **Question 2:** The subsumption hierarcy is defined for fully satisfied criteria (i.e., 100% coverage). Does it hold for any degree of coverage? For example, does 80% edge coverage imply at least 80% node coverage?