Graph-Based Adequacy Criteria

"Testers find a graph and cover it"
- Boris Beizer

Graphs

- Intuitive ways of expressing specifications and code.
  - Consequently, graph representations are commonly used in structural testing
- Graph-based ***adequacy criteria***
  - …state how graph representations of the implementation should be covered during testing
- Graph-based ***coverage***
  - …measures to what extent the graph abstractions have been covered

Example: A Simple Program

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

  for (i = 1; i <= n; ++i) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (k = 1; k <= n; ++k) {
    mini = -1;
    for (i = 1; i <= n; ++i)
      if (!visited[i] && ((mini == -1) || (d[i] < d[mini])))
        mini = i;

    visited[mini] = 1;
    for (i = 1; i <= n; ++i)
      if (dist[mini][i])
        if (d[mini] + dist[mini][i] < d[i])
          d[i] = d[mini] + dist[mini][i];
  }
}
```

Graph Representation

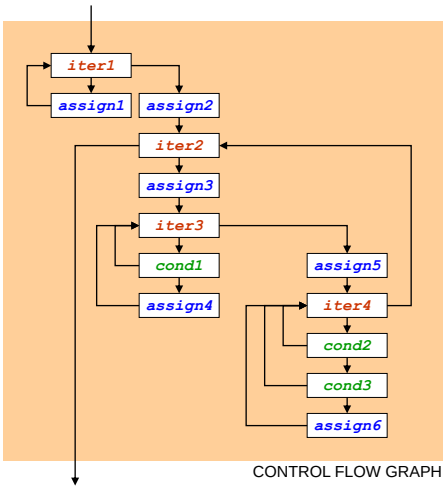
```
void dijkstra(int s) {
  int i, k, mini;
  int visited[GRAPHSIZE];

  for (i = 1; i <= n; ++i) {
    d[i] = INFINITY;
    visited[i] = 0;
  }

  d[s] = 0;

  for (k = 1; k <= n; ++k) {
    mini = -1;
    for (i = 1; i <= n; ++i)
      if (cond1)
        mini = i;

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

Graph Representation

```
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];
  }
}
```

Graph Representation

```
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
  }
}
```



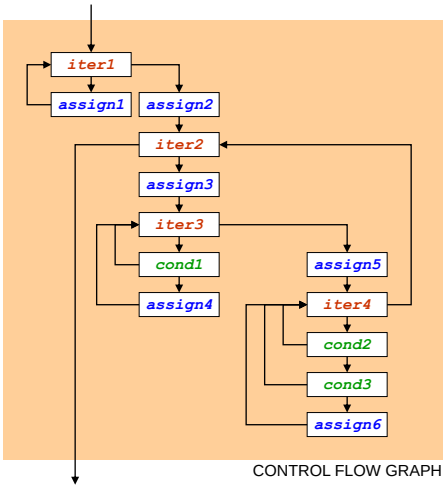
CONTROL FLOW GRAPH

Control Flow Adequacy Criteria

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

  for (iter1) {
    assign1
  }

  assign2

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

    assign5
  }
}
```

Adequacy Criterion

- **Statement Coverage**



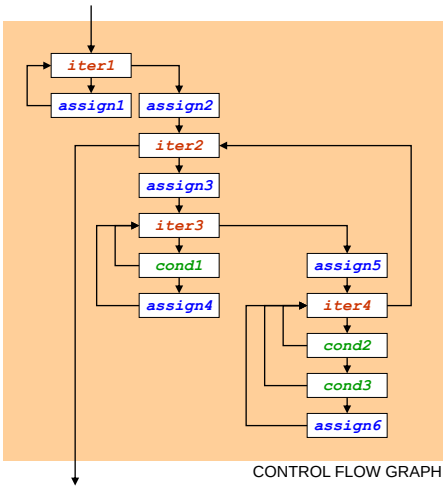
CONTROL FLOW GRAPH

Control Flow Adequacy Criteria

```
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
  }
}
```

Adequacy Criterion

- Statement Coverage
- **Edge Coverage**



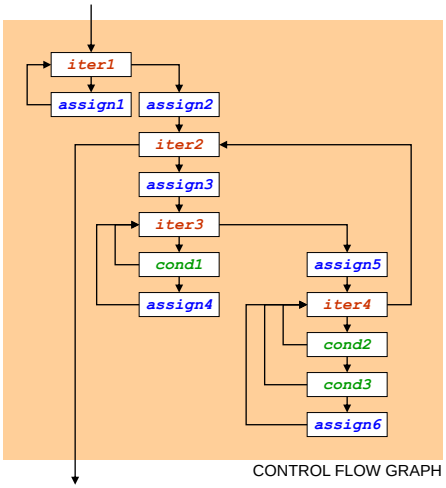
CONTROL FLOW GRAPH

Control Flow Adequacy Criteria

```
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
  }
}
```

Adequacy Criterion

- Statement Coverage
- Edge Coverage
- **Path Coverage**



CONTROL FLOW GRAPH