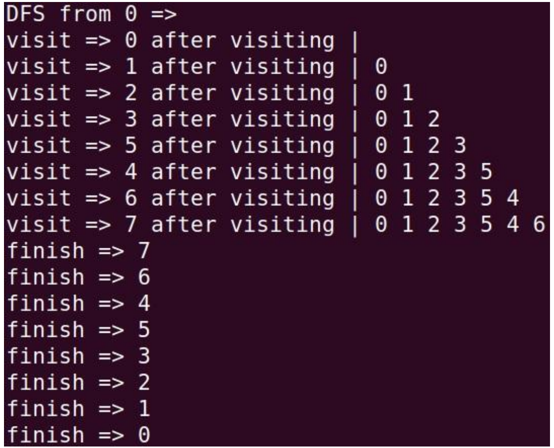
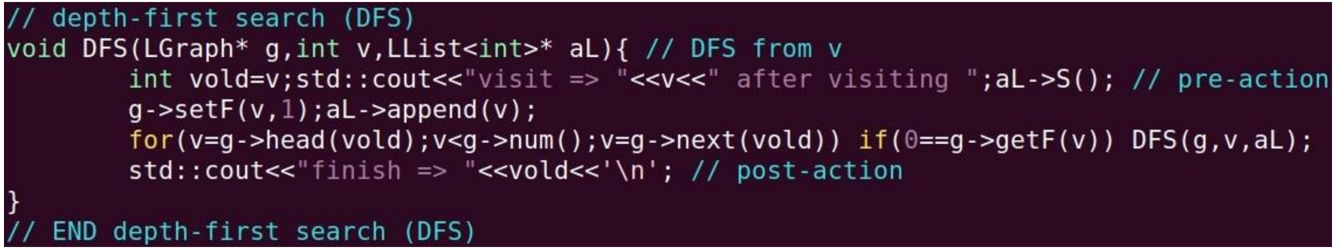
Graph

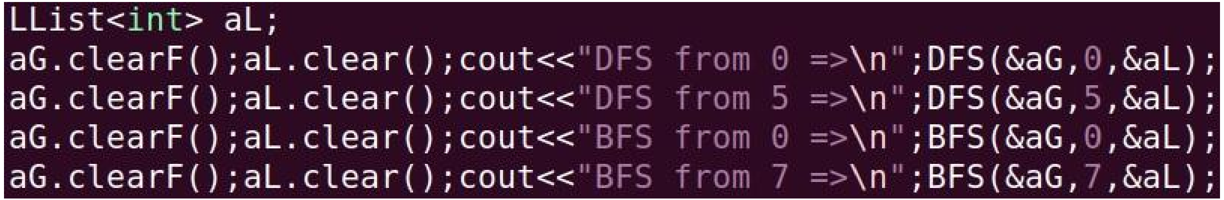
- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours



```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



```
// depth-first search (DFS)
void DFS(LGraph* g,int v,LList<int>* aL){ // DFS from v
        int vold=v;std::cout<<"visit => "<<v<<" after visiting ";aL->S(); // pre-action
        g->setF(v,1);aL->append(v);
        for(v=g->head(vold);v<g->num();v=g->next(vold)) if(0==g->getF(v)) DFS(g,v,aL);
        std::cout<<"finish => "<<vold<<'\n'; // post-action
}
// END depth-first search (DFS)
```

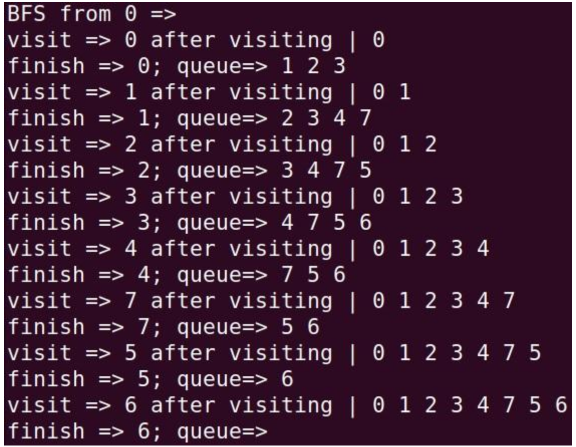


```
LList<int> aL;
aG.clearF();aL.clear();cout<<"DFS from 0 =>\n";DFS(&aG,0,&aL);
aG.clearF();aL.clear();cout<<"DFS from 5 =>\n";DFS(&aG,5,&aL);
aG.clearF();aL.clear();cout<<"BFS from 0 =>\n";BFS(&aG,0,&aL);
aG.clearF();aL.clear();cout<<"BFS from 7 =>\n";BFS(&aG,7,&aL);
```

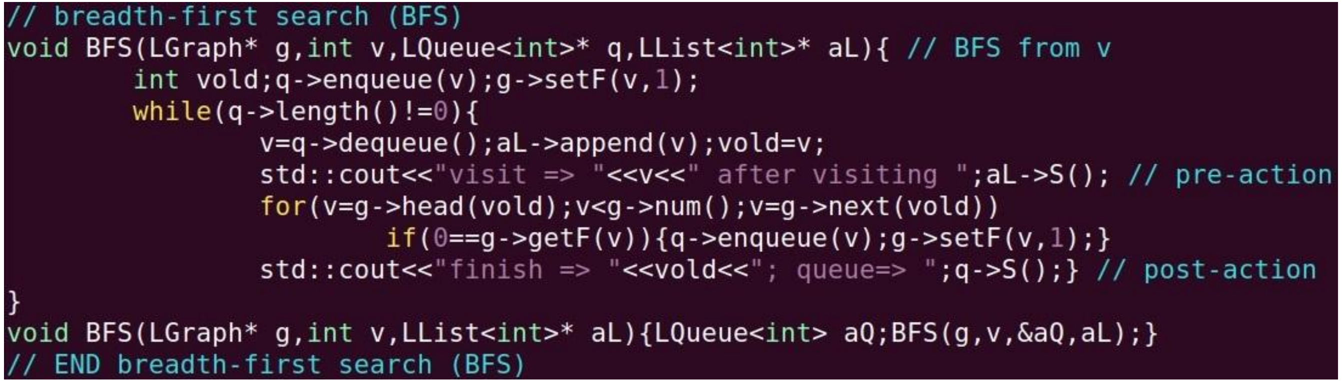
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - *depth first search* (DFS)
    - visit a vertex after visiting all its neighbours
  - ***breadth first search*** (BFS)
    - visit a vertex after visiting all its neighbours

take advantage of a *queue*
that performs FIFO (first in first out)



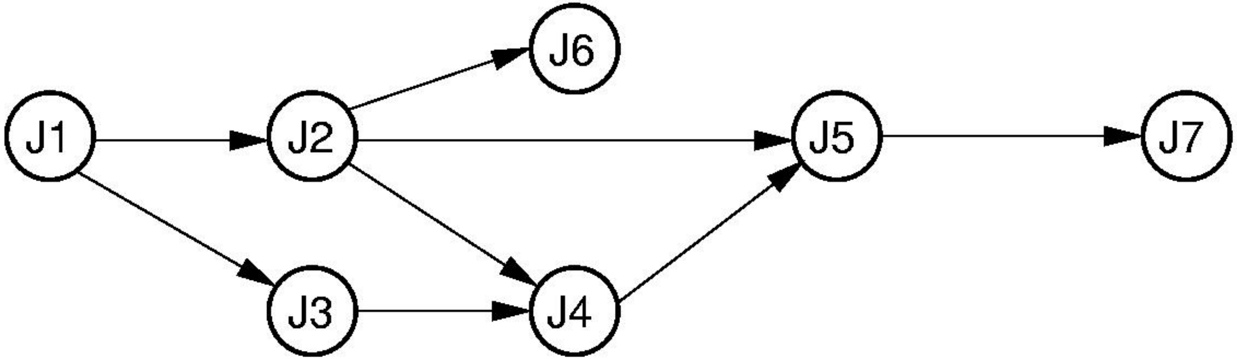
```
BFS from 0 =>
visit => 0 after visiting | 0
finish => 0; queue=> 1 2 3
visit => 1 after visiting | 0 1
finish => 1; queue=> 2 3 4 7
visit => 2 after visiting | 0 1 2
finish => 2; queue=> 3 4 7 5
visit => 3 after visiting | 0 1 2 3
finish => 3; queue=> 4 7 5 6
visit => 4 after visiting | 0 1 2 3 4
finish => 4; queue=> 7 5 6
visit => 7 after visiting | 0 1 2 3 4 7
finish => 7; queue=> 5 6
visit => 5 after visiting | 0 1 2 3 4 7 5
finish => 5; queue=> 6
visit => 6 after visiting | 0 1 2 3 4 7 5 6
finish => 6; queue=>
```



```
// breadth-first search (BFS)
void BFS(LGraph* g,int v,LQueue<int>* q,LList<int>* aL){ // BFS from v
        int vold;q->enqueue(v);g->setF(v,1);
        while(q->length()!=0){
                v=q->dequeue();aL->append(v);vold=v;
                std::cout<<"visit => "<<v<<" after visiting ";aL->S(); // pre-action
                for(v=g->head(vold);v<g->num();v=g->next(vold))
                        if(0==g->getF(v)){q->enqueue(v);g->setF(v,1);}
                std::cout<<"finish => "<<vold<<"; queue=> ";q->S();} // post-action
}
void BFS(LGraph* g,int v,LList<int>* aL){LQueue<int> aQ;BFS(g,v,&aQ,aL);}
// END breadth-first search (BFS)
```

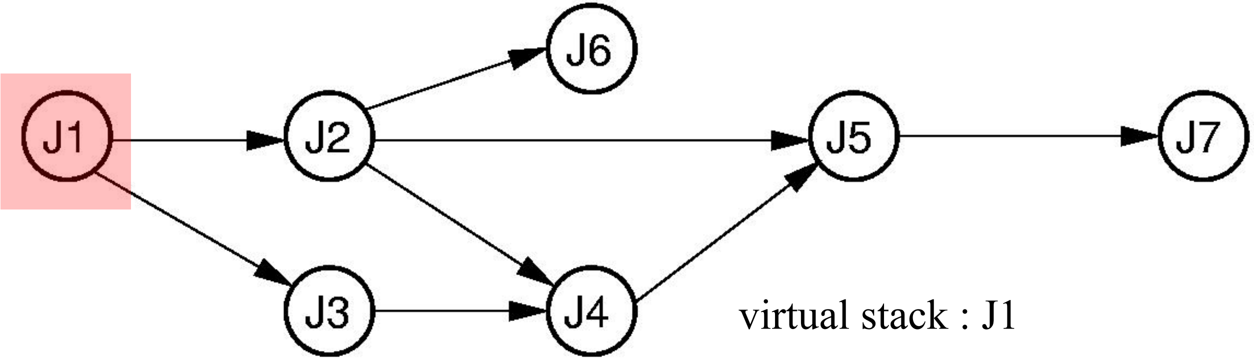
Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - *breadth first search* (BFS)



Graph

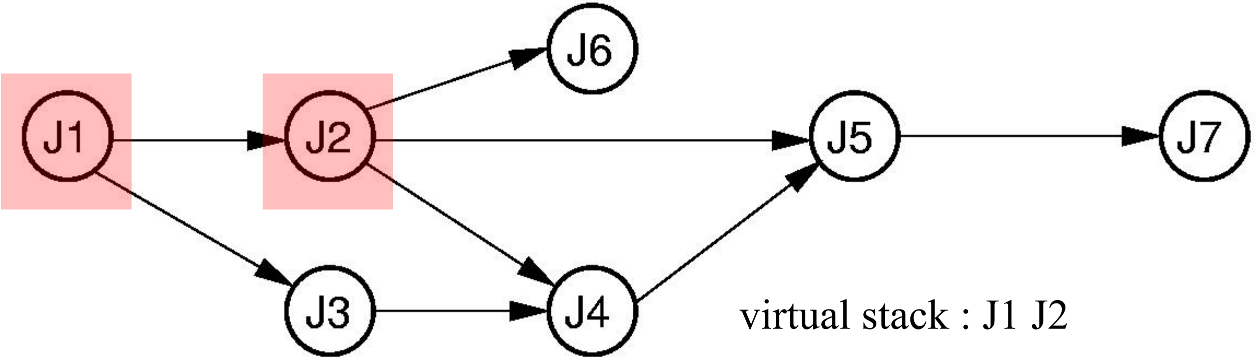
- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)



virtual stack : J1

Graph

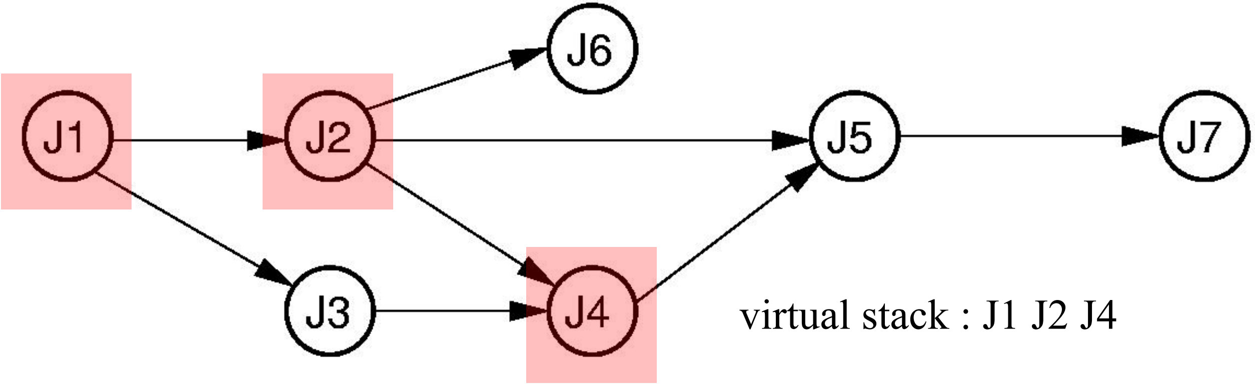
- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)



virtual stack : J1 J2

Graph

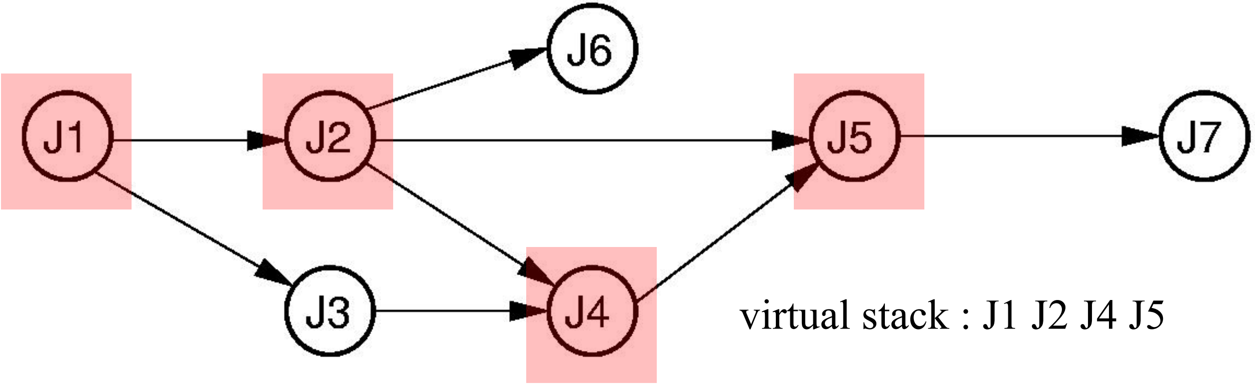
- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)



virtual stack : J1 J2 J4

Graph

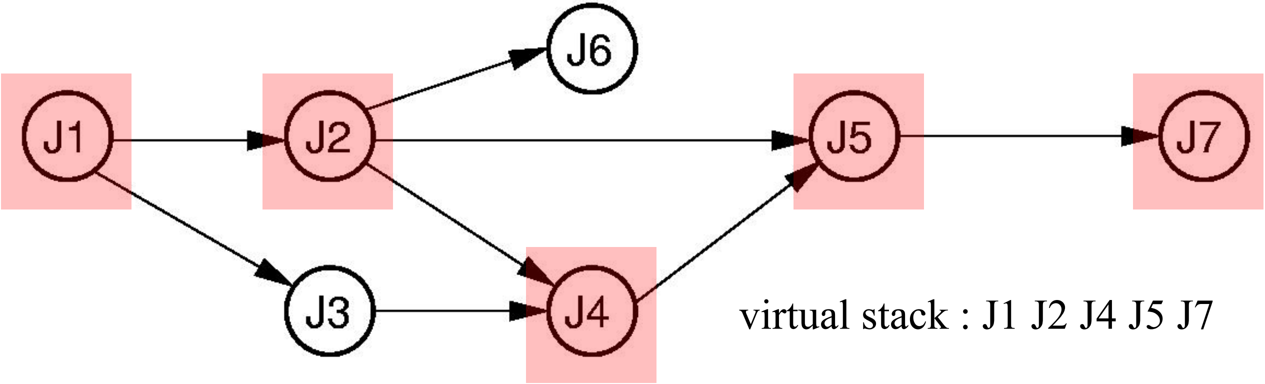
- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)



virtual stack : J1 J2 J4 J5

Graph

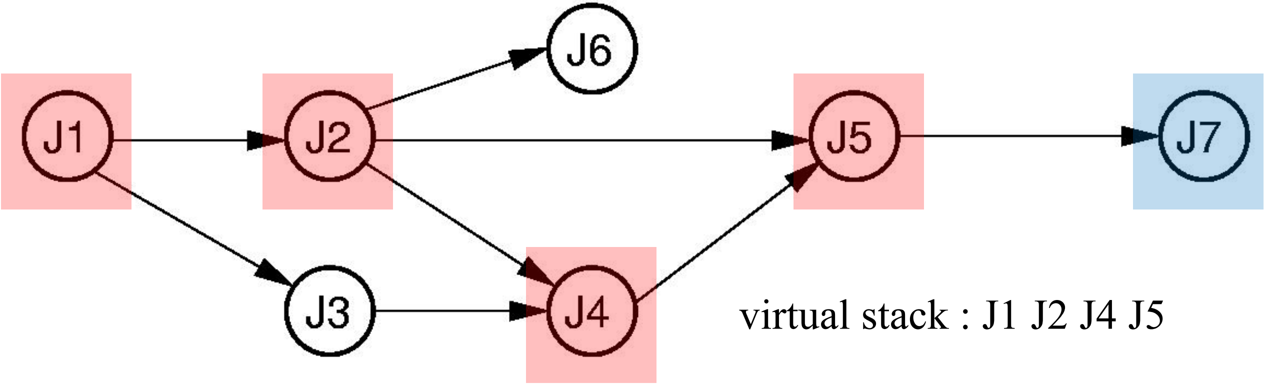
- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)



virtual stack : J1 J2 J4 J5 J7

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)

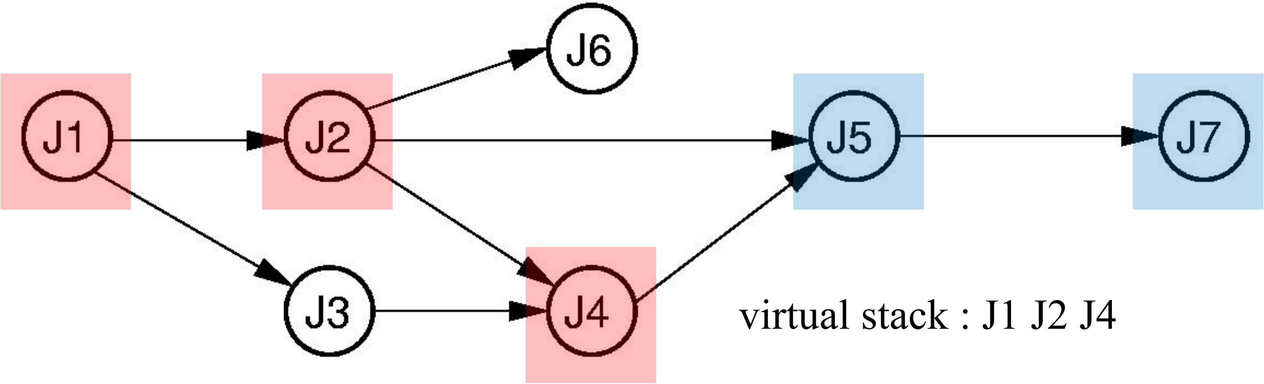


virtual stack : J1 J2 J4 J5

J7

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)

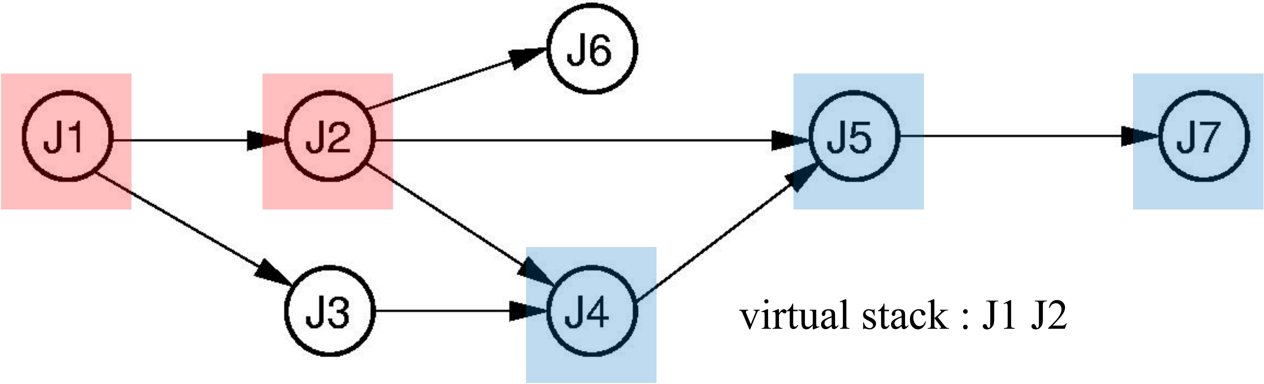


virtual stack : J1 J2 J4

J5 => J7

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)

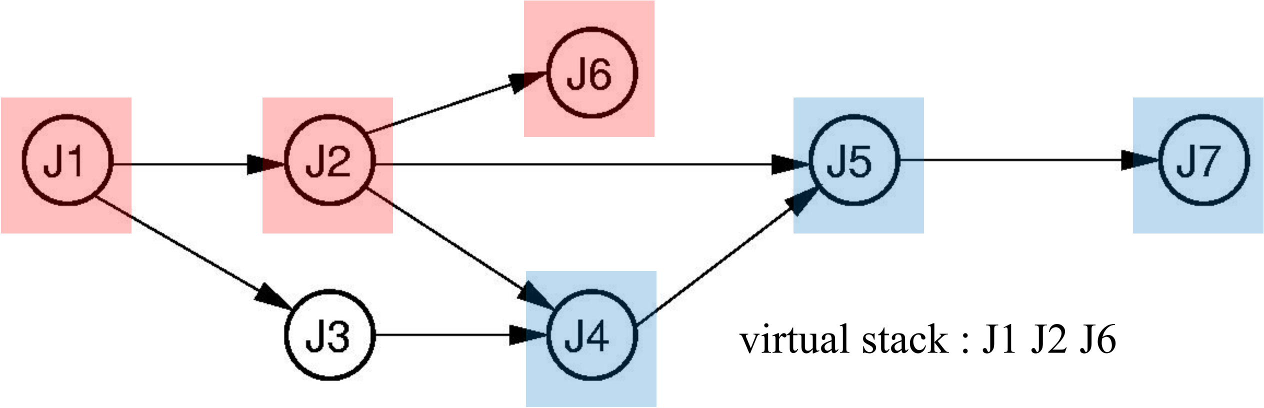


virtual stack : J1 J2

J4 => J5 => J7

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)

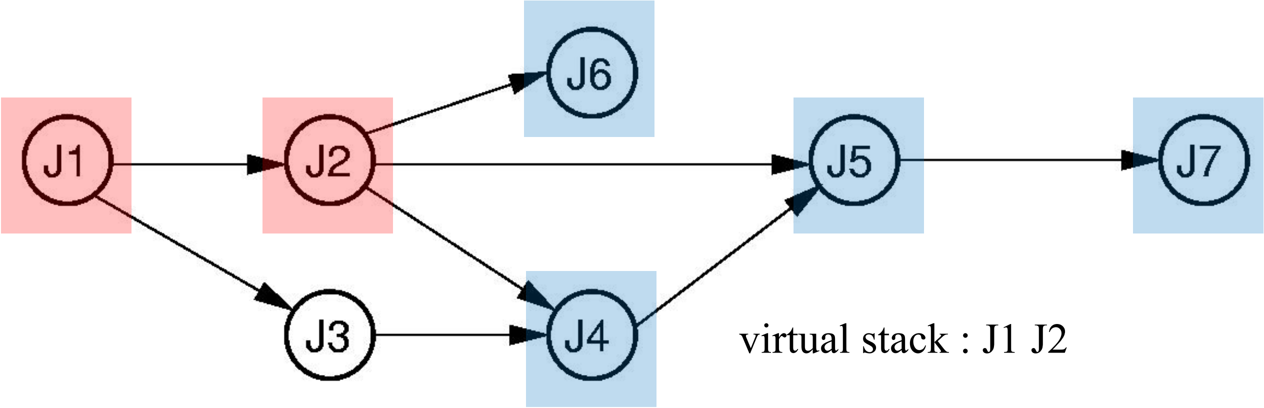


virtual stack : J1 J2 J6

J4 => J5 => J7

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)

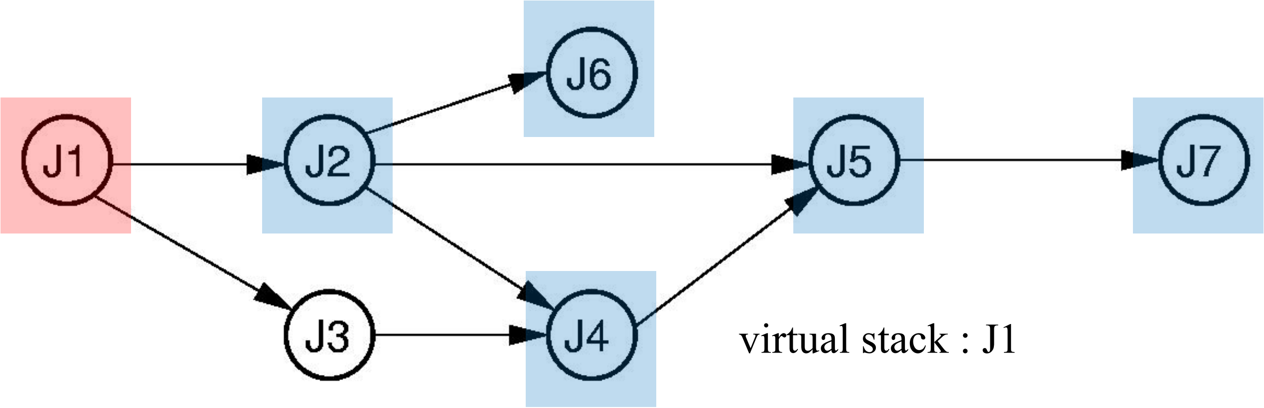


virtual stack : J1 J2

J6 => J4 => J5 => J7

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)

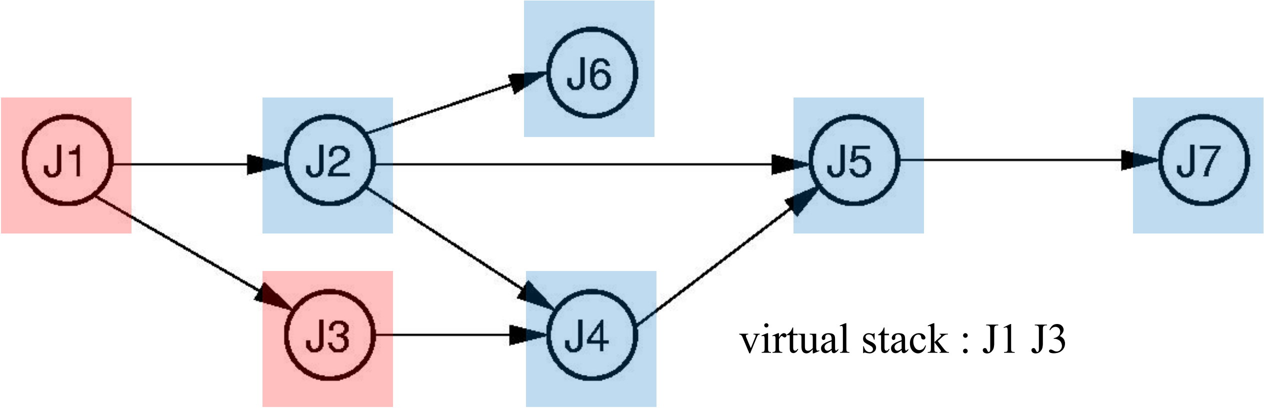


virtual stack : J1

J2 => J6 => J4 => J5 => J7

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)

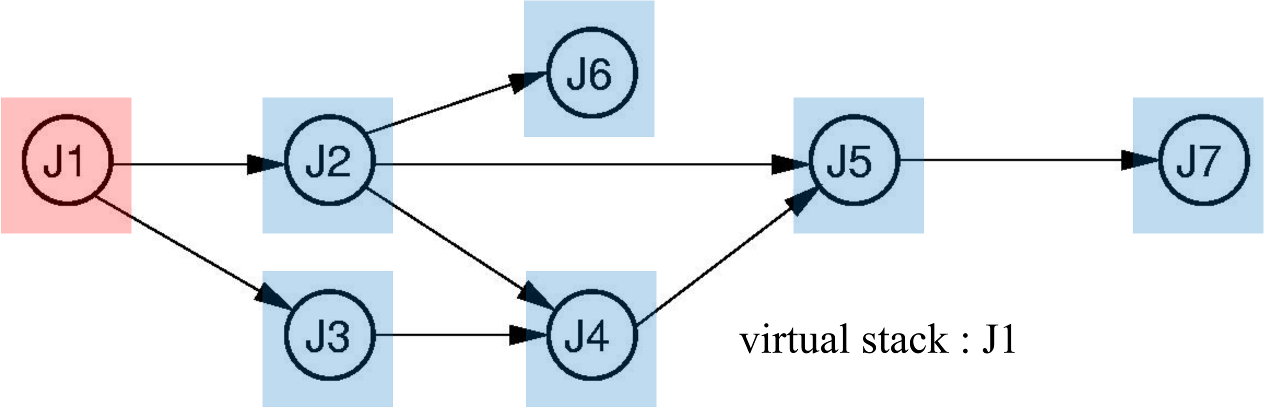


virtual stack : J1 J3

J2 => J6 => J4 => J5 => J7

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)

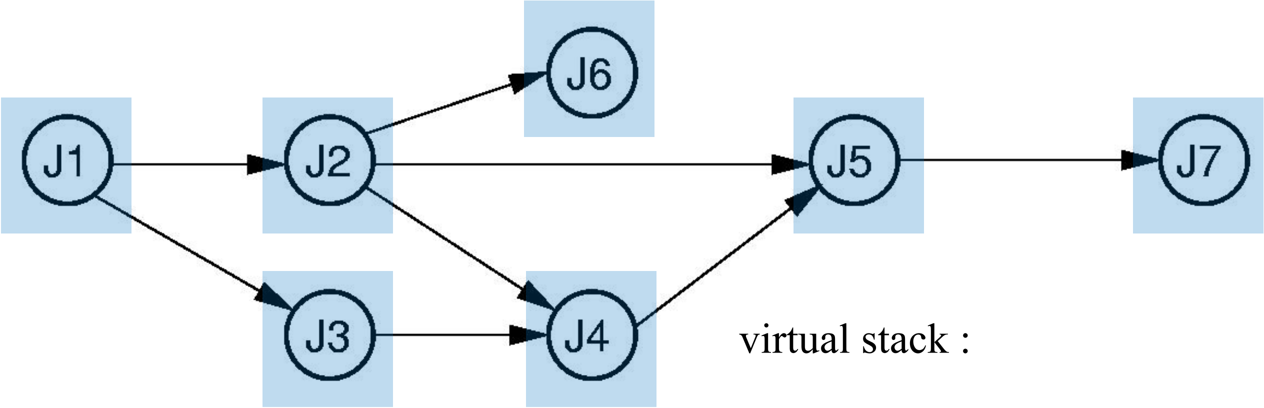


virtual stack : J1

J3 => J2 => J6 => J4 => J5 => J7

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - ***depth first search*** (DFS)
  - *breadth first search* (BFS)

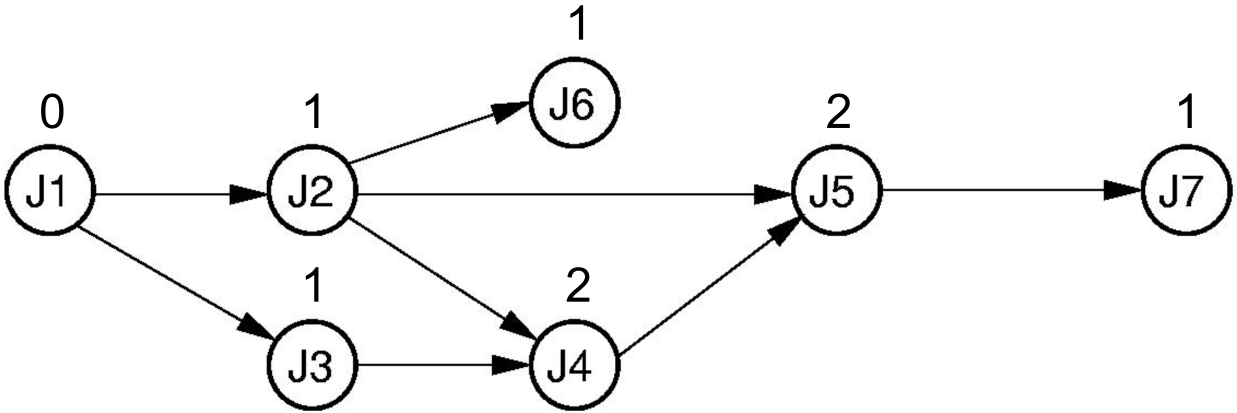


virtual stack :

J1 => J3 => J2 => J6 => J4 => J5 => J7

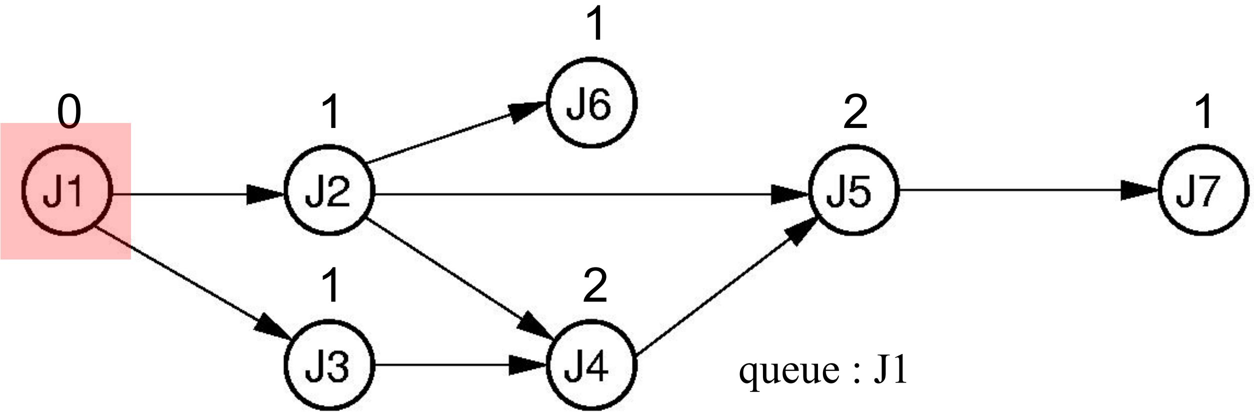
Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update



Graph

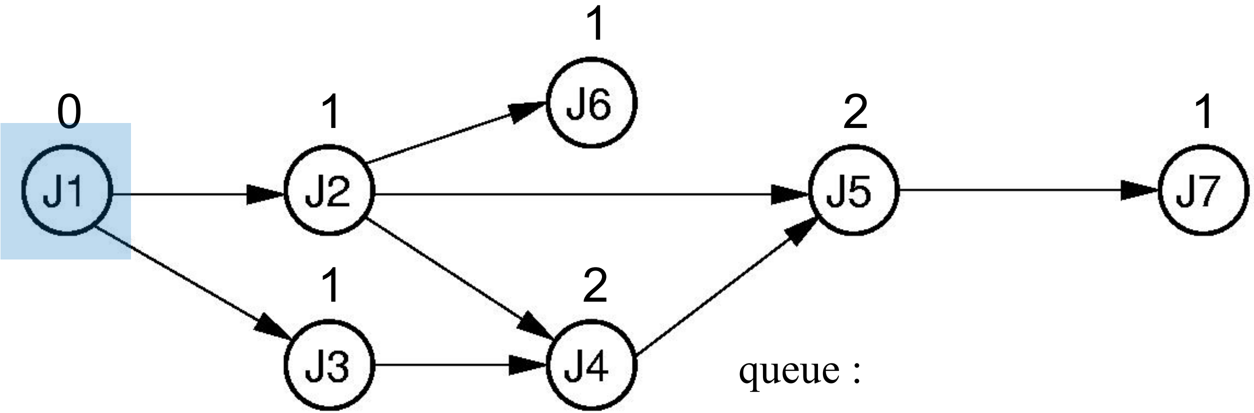
- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update



queue : J1

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

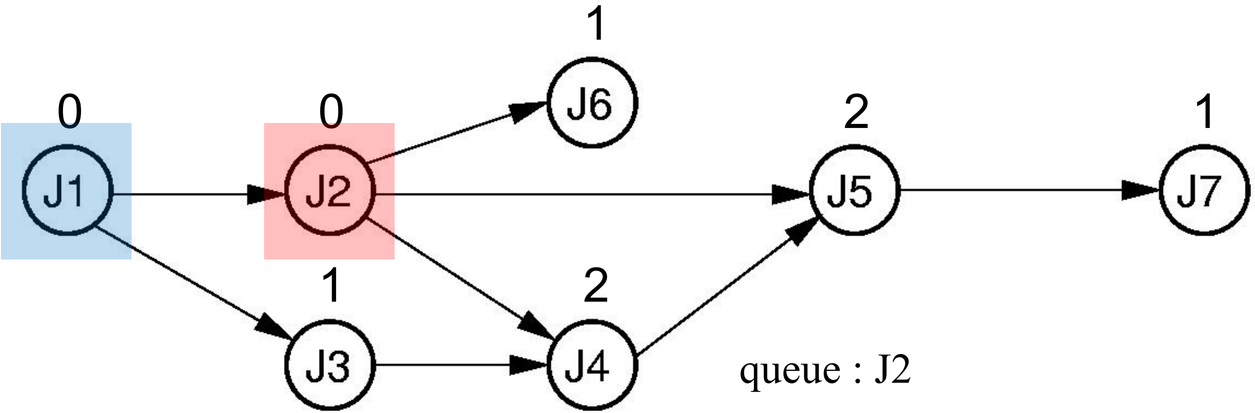


queue :

J1

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

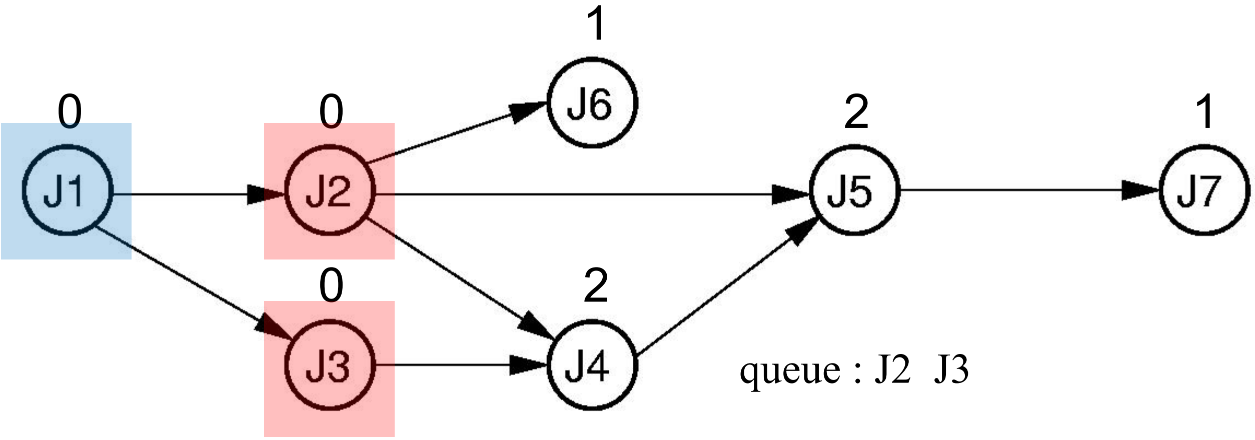


queue : J2

J1

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

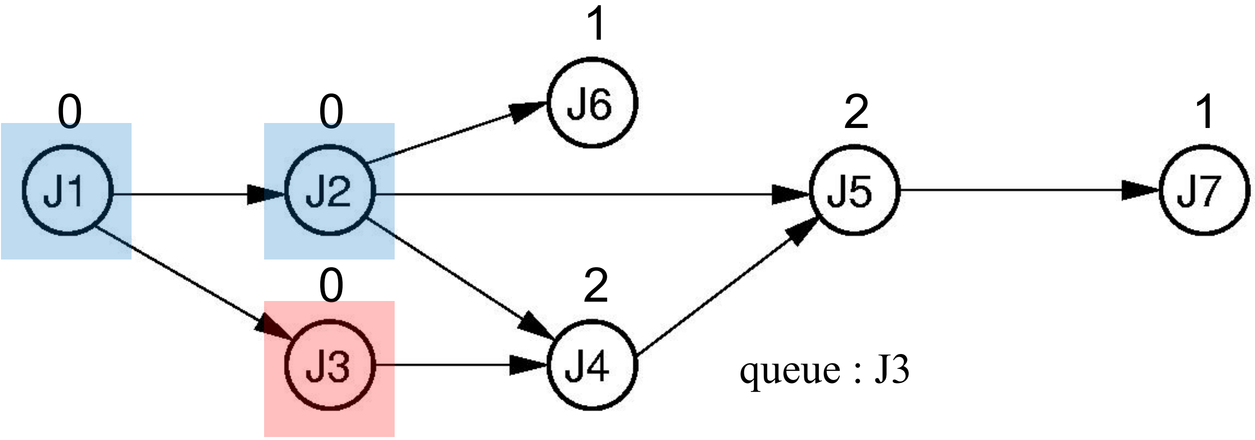


queue : J2 J3

J1

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

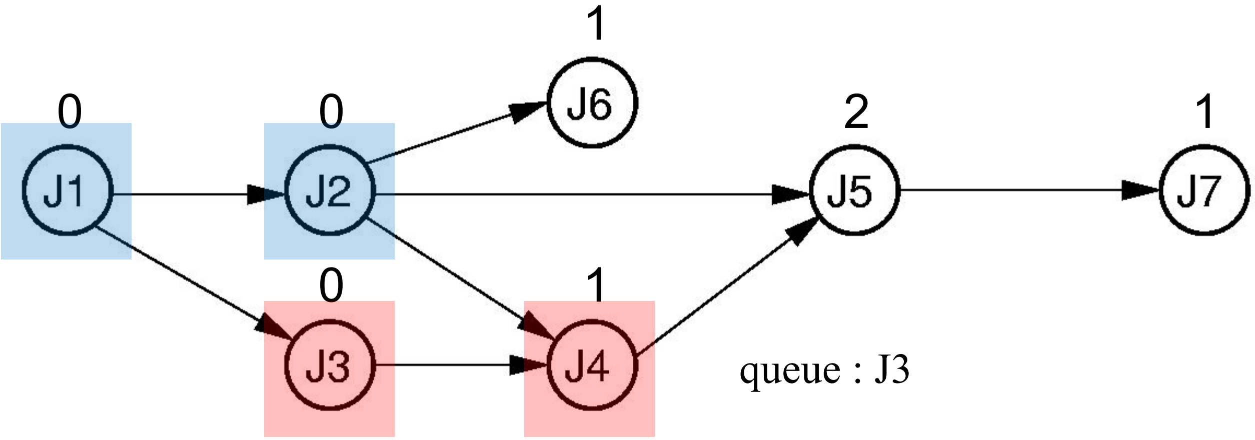


queue : J3

J1 => J2

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

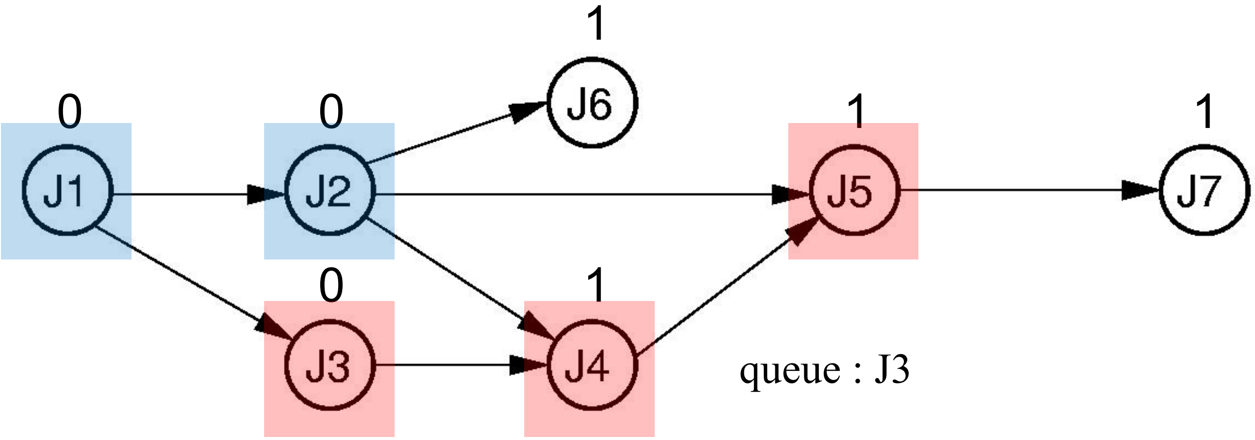


queue : J3

J1 => J2

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

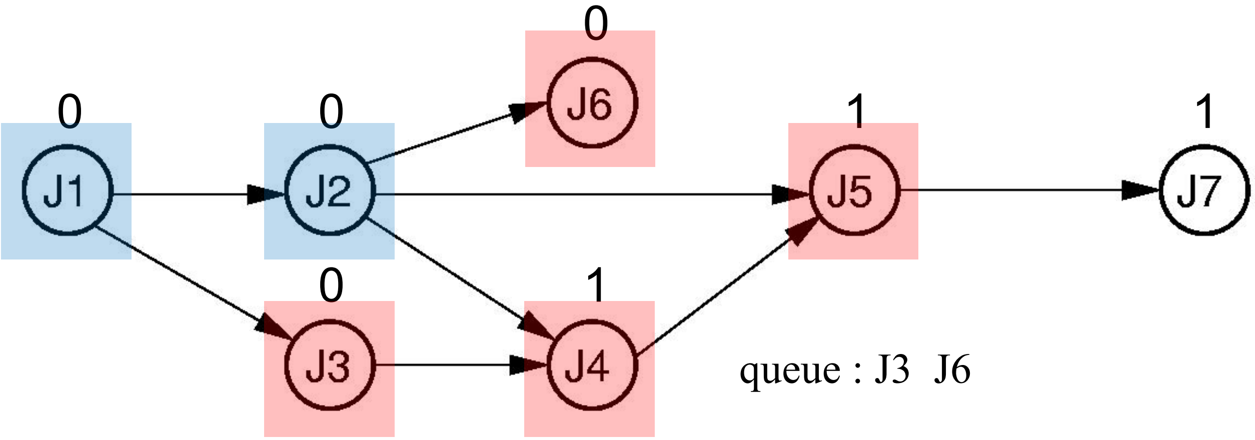


queue : J3

J1 => J2

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

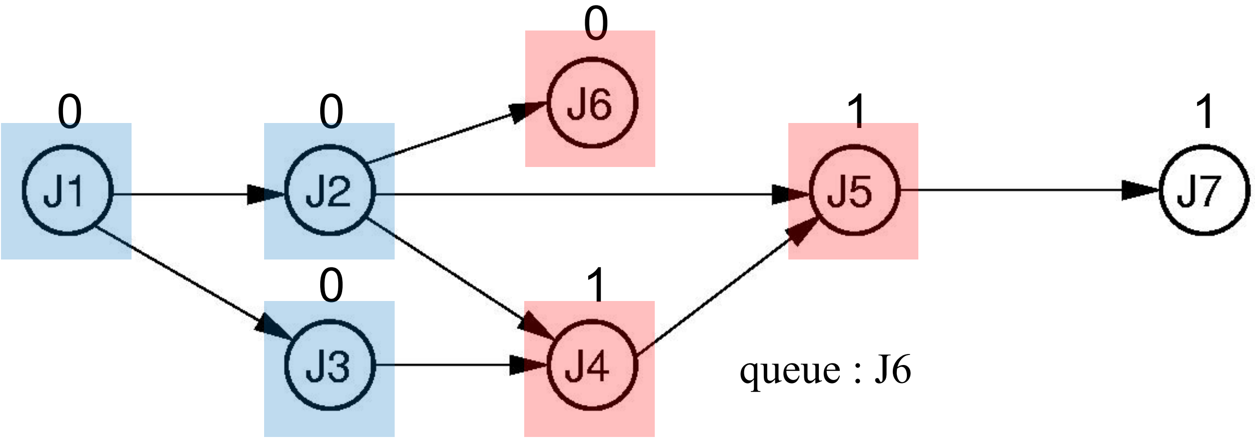


queue : J3 J6

J1 => J2

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

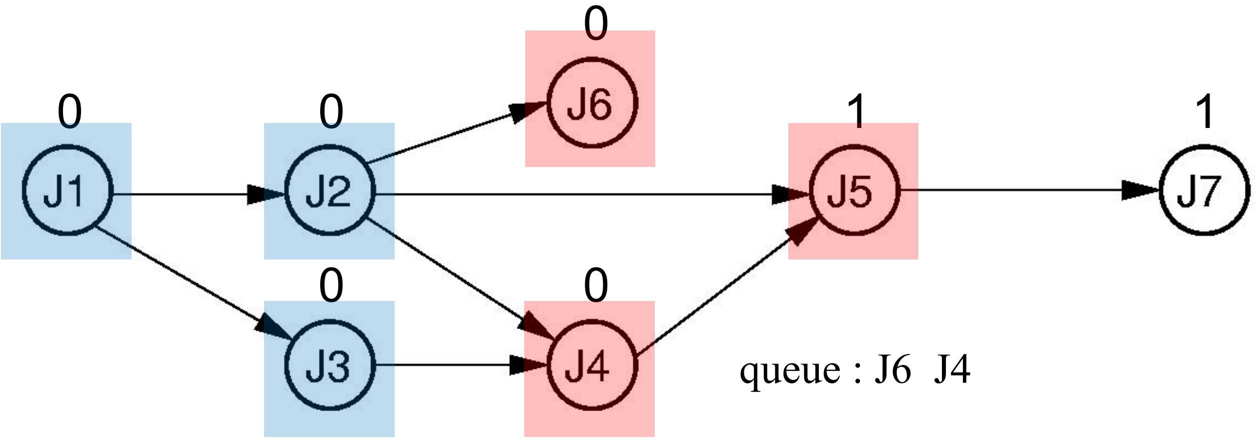


queue : J6

J1 => J2 => J3

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

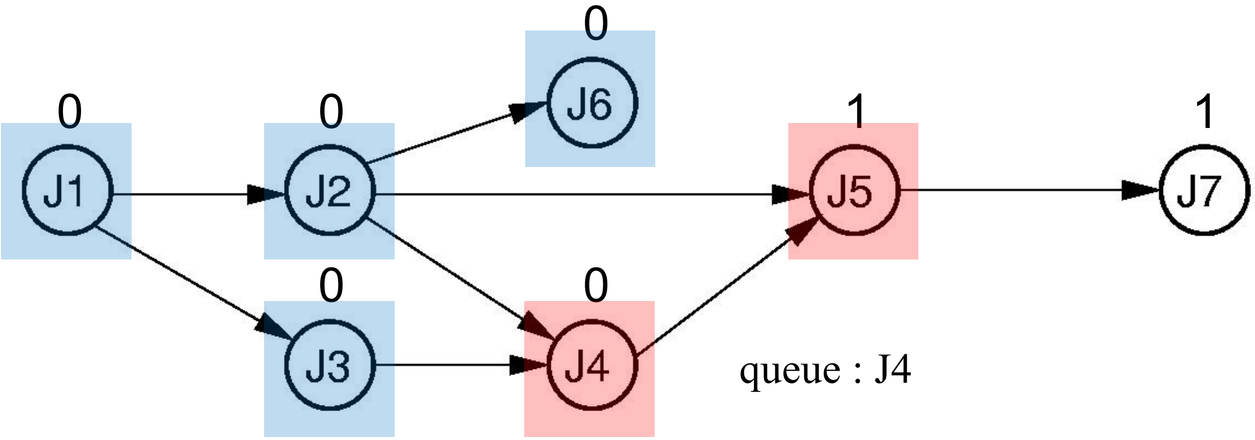


queue : J6 J4

J1 => J2 => J3

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

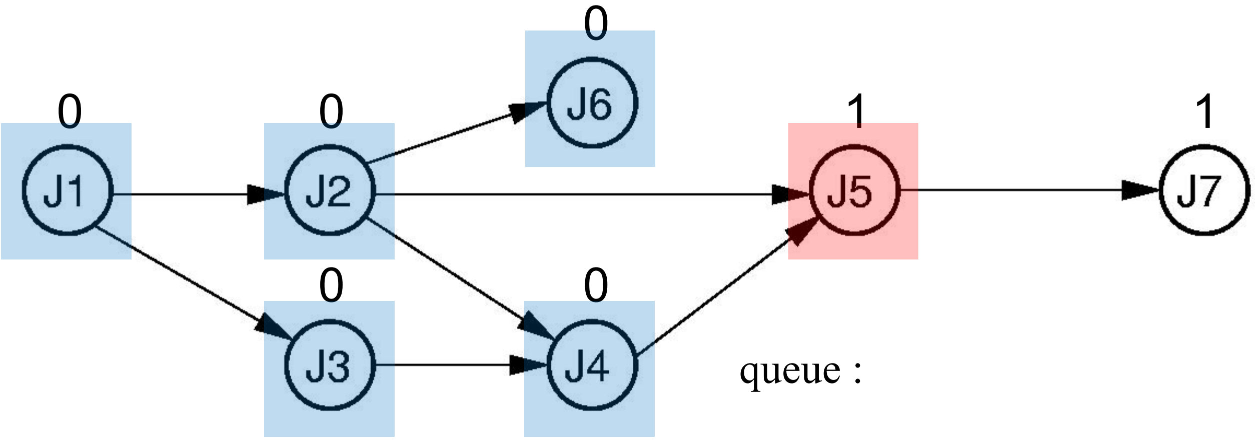


queue : J4

J1 => J2 => J3 => J6

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

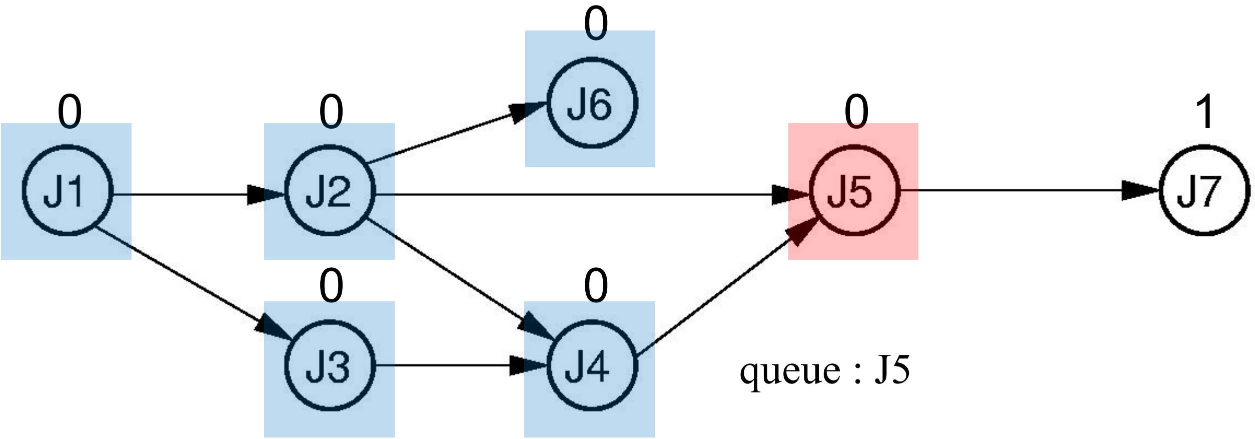


queue :

J1 => J2 => J3 => J6 => J4

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

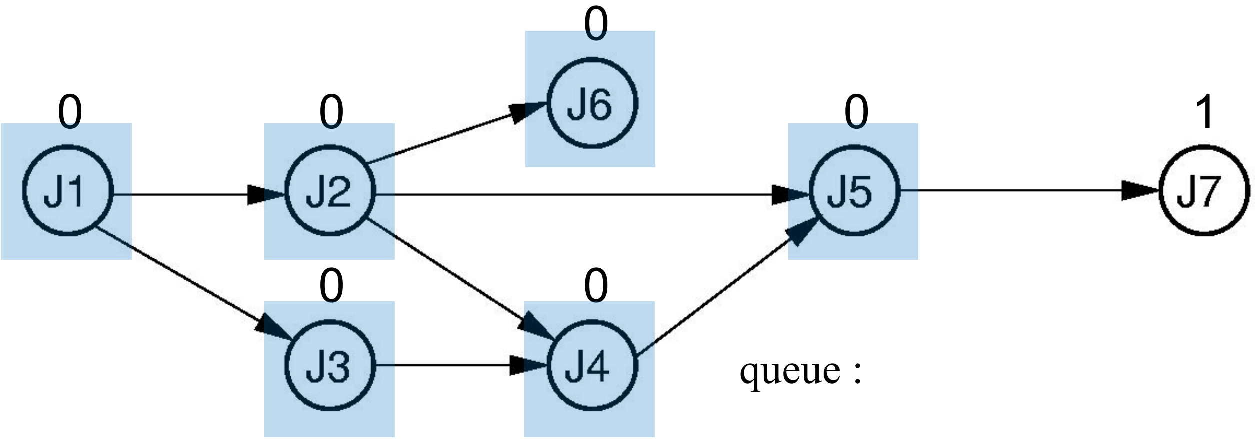


queue : J5

J1 => J2 => J3 => J6 => J4

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

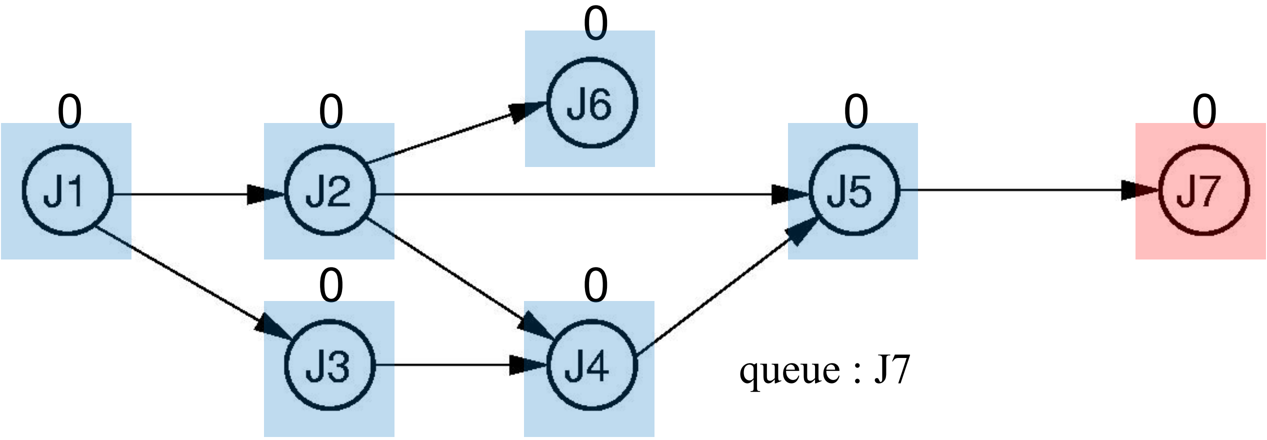


queue :

J1 => J2 => J3 => J6 => J4 => J5

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update

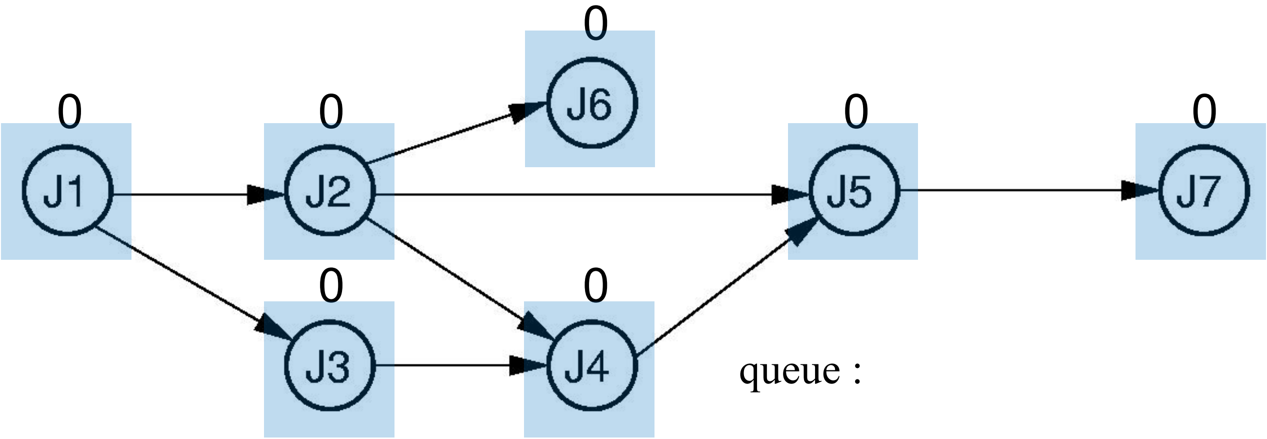


queue : J7

J1 => J2 => J3 => J6 => J4 => J5

Graph

- **Graph traversal - topological sort**
  - Given a set of jobs with prerequisites, order the jobs without violating prerequisites
  - *depth first search* (DFS)
  - ***breadth first search*** (BFS) - dynamic *in degree* update



queue :

J1 => J2 => J3 => J6 => J4 => J5 => J7

THANK YOU

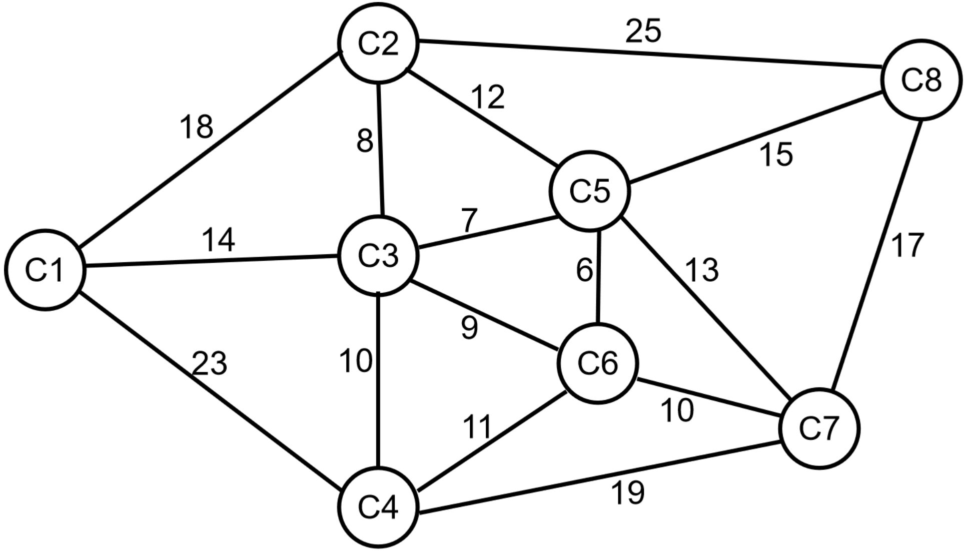
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

	C1	C2	C3	C4	C5	C6	C7	C8
C1	0	18	14	23	21	23	33	36
C2	18	0	8	18	12	17	25	25
C3	14	8	0	10	7	9	19	22
C4	23	18	10	0	17	11	19	32
C5	21	12	7	17	0	6	13	15
C6	23	17	9	11	6	0	10	21
C7	33	25	19	19	13	10	0	17
C8	36	25	22	32	15	21	17	0

Graph

- **Graph - shortest paths**
  - directed graph G=(V,E)
  - *Dijkstra* algorithm - single-pair shortest path
  - *Floyd* algorithm - all-pairs shortest paths

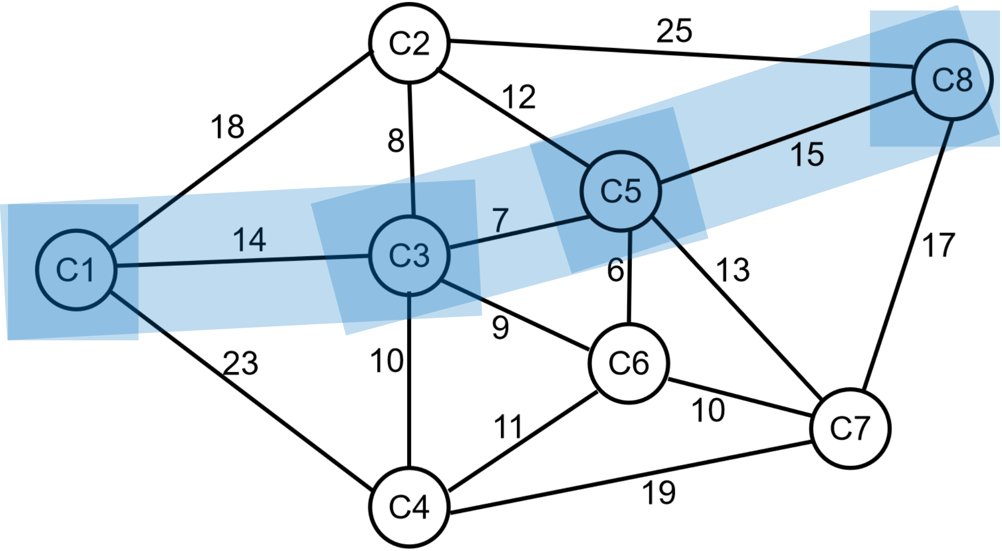
min-distance(C1,C8) = 36
min-path(C1,C8): C1=>C3=>C5=>C8



Graph

- **Graph - shortest paths**
  - ***Dijkstra* algorithm** - single-pair shortest path

min-distance(C1,C8) = 36
min-path(C1,C8): C1=>C3=>C5=>C8

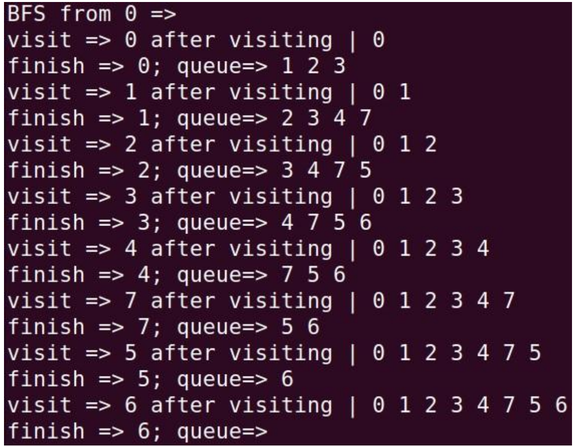


Graph

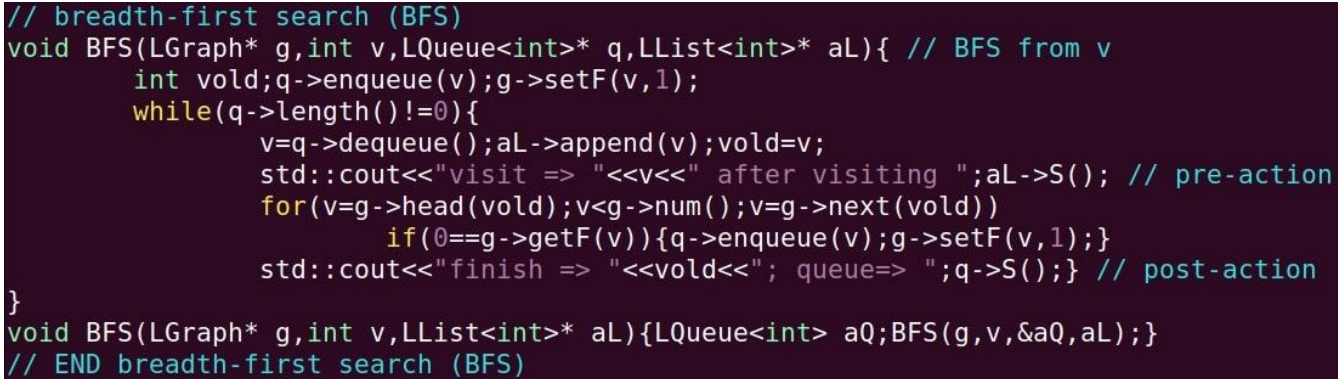
REVIEW

- **Graph traversal**
  - directed graph G=(V,E)
  - *depth first search* (DFS)
    - visit a vertex after visiting all its neighbours
  - ***breadth first search*** (BFS)
    - visit a vertex after visiting all its neighbours

BFS relies on a *queue*
that performs FIFO (first in first out)



```
BFS from 0 =>
visit => 0 after visiting | 0
finish => 0; queue=> 1 2 3
visit => 1 after visiting | 0 1
finish => 1; queue=> 2 3 4 7
visit => 2 after visiting | 0 1 2
finish => 2; queue=> 3 4 7 5
visit => 3 after visiting | 0 1 2 3
finish => 3; queue=> 4 7 5 6
visit => 4 after visiting | 0 1 2 3 4
finish => 4; queue=> 7 5 6
visit => 7 after visiting | 0 1 2 3 4 7
finish => 7; queue=> 5 6
visit => 5 after visiting | 0 1 2 3 4 7 5
finish => 5; queue=> 6
visit => 6 after visiting | 0 1 2 3 4 7 5 6
finish => 6; queue=>
```

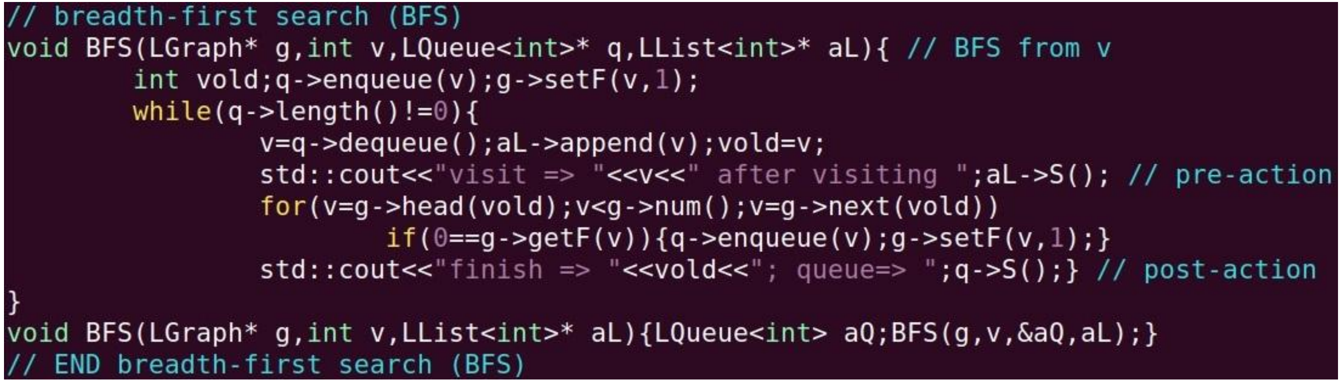


```
// breadth-first search (BFS)
void BFS(LGraph* g,int v,LQueue<int>* q,LList<int>* aL){ // BFS from v
        int vold;q->enqueue(v);g->setF(v,1);
        while(q->length()!=0){
                v=q->dequeue();aL->append(v);vold=v;
                std::cout<<"visit => "<<v<<" after visiting ";aL->S(); // pre-action
                for(v=g->head(vold);v<g->num();v=g->next(vold))
                        if(0==g->getF(v)){q->enqueue(v);g->setF(v,1);}
                std::cout<<"finish => "<<vold<<"; queue=> ";q->S();} // post-action
}
void BFS(LGraph* g,int v,LList<int>* aL){LQueue<int> aQ;BFS(g,v,&aQ,aL);}
// END breadth-first search (BFS)
```

Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

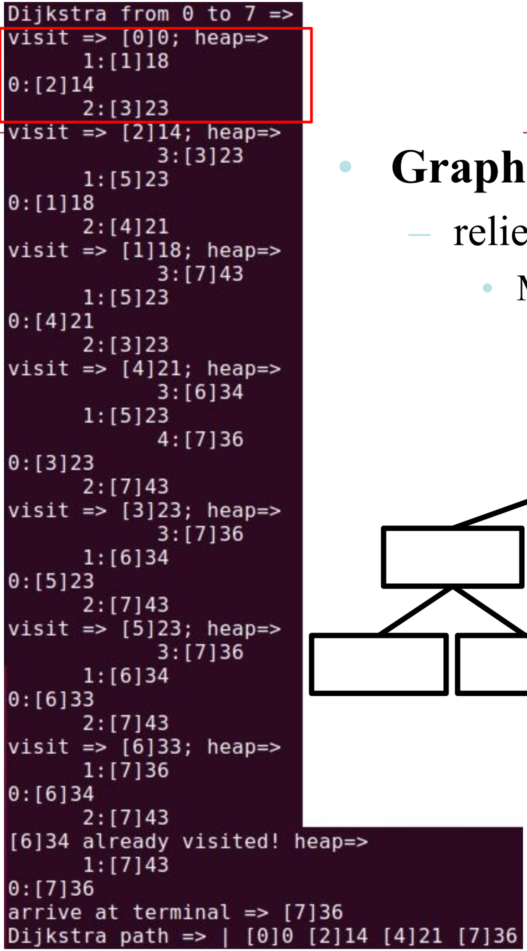
BFS relies on a *queue*
that performs FIFO (first in first out)



```
// breadth-first search (BFS)
void BFS(LGraph* g,int v,LQueue<int>* q,LList<int>* aL){ // BFS from v
        int vold;q->enqueue(v);g->setF(v,1);
        while(q->length()!=0){
                v=q->dequeue();aL->append(v);vold=v;
                std::cout<<"visit => "<<v<<" after visiting ";aL->S(); // pre-action
                for(v=g->head(vold);v<g->num();v=g->next(vold))
                        if(0==g->getF(v)){q->enqueue(v);g->setF(v,1);}
                std::cout<<"finish => "<<vold<<"; queue=> ";q->S();} // post-action
}
void BFS(LGraph* g,int v,LList<int>* aL){LQueue<int> aQ;BFS(g,v,&aQ,aL);}
// END breadth-first search (BFS)
```

Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)
  - *sparse graphs* normally resort to a *heap* for efficient MDFO implementation
    - sparse graphs are much more common than dense graphs in practical applications
    - sparse graphs normally adopt adjacency list representation
    - heap based MDFO is dedicated to sparse graphs that adopt adjacency list representation
  - *dense graphs* normally resort to *direct vertex traversal* for MDFO implementation
    - heap based MDFO brings no benefit to dense graphs
    - dense graphs are much less common than sparse graphs in practical applications

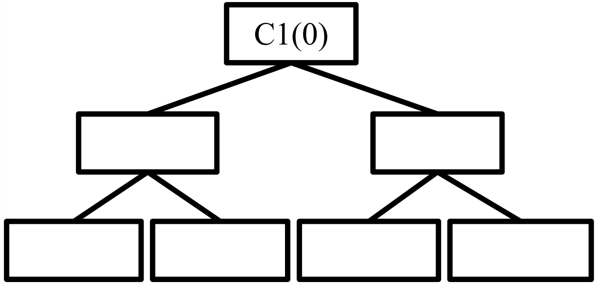


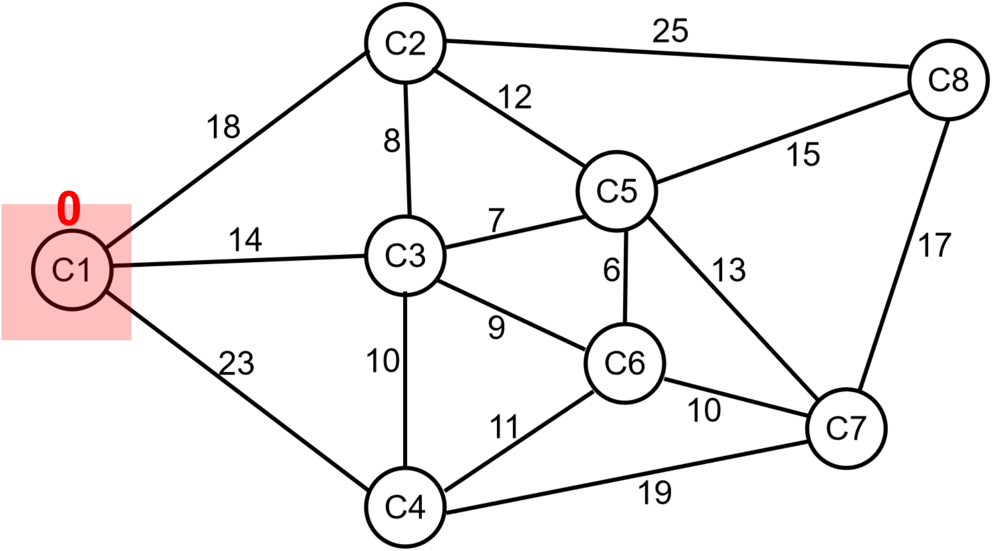
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
        1:[1]18
0:[2]14
        2:[3]23
visit => [2]14; heap=>
                3:[3]23
        1:[5]23
0:[1]18
        2:[4]21
visit => [1]18; heap=>
                3:[7]43
        1:[5]23
0:[4]21
        2:[3]23
visit => [4]21; heap=>
                3:[6]34
        1:[5]23
                4:[7]36
0:[3]23
        2:[7]43
visit => [3]23; heap=>
                3:[7]36
        1:[6]34
0:[5]23
        2:[7]43
visit => [5]23; heap=>
                3:[7]36
        1:[6]34
0:[6]33
        2:[7]43
visit => [6]33; heap=>
        1:[7]36
0:[6]34
        2:[7]43
[6]34 already visited! heap=>
        1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

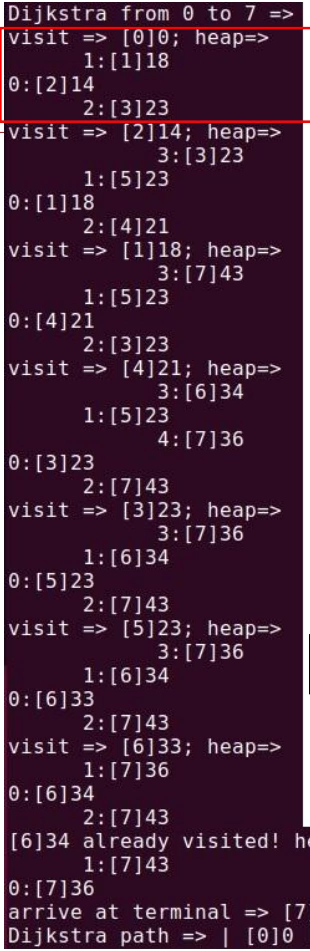
Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap : C1





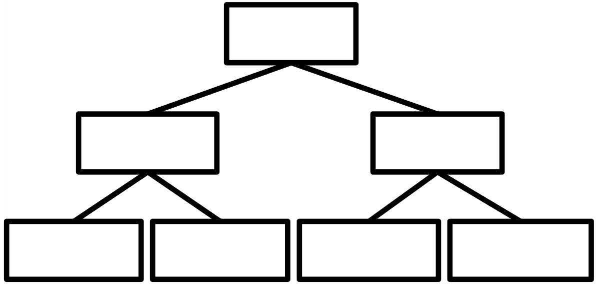


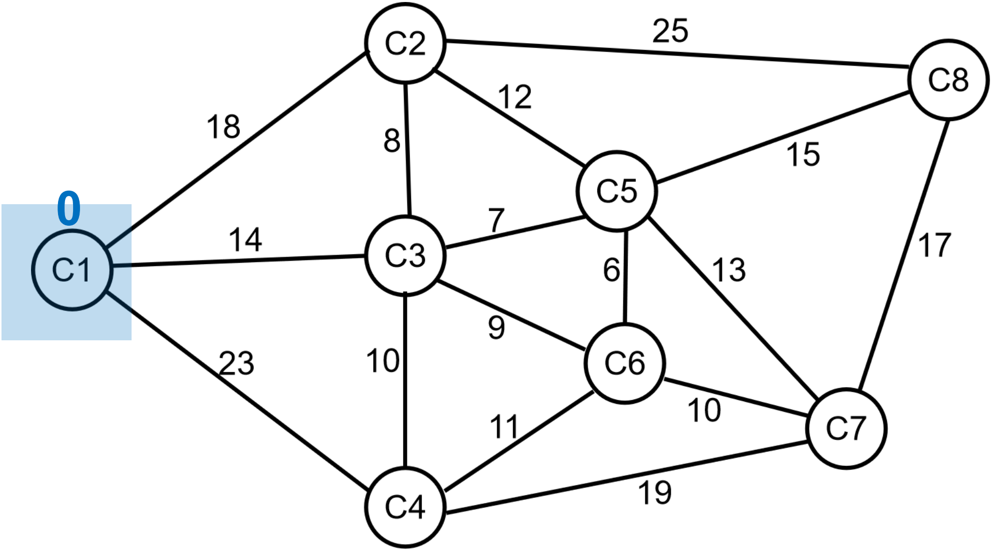
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
        1:[1]18
0:[2]14
        2:[3]23
visit => [2]14; heap=>
                3:[3]23
        1:[5]23
0:[1]18
        2:[4]21
visit => [1]18; heap=>
                3:[7]43
        1:[5]23
0:[4]21
        2:[3]23
visit => [4]21; heap=>
                3:[6]34
        1:[5]23
                4:[7]36
0:[3]23
        2:[7]43
visit => [3]23; heap=>
                3:[7]36
        1:[6]34
0:[5]23
        2:[7]43
visit => [5]23; heap=>
                3:[7]36
        1:[6]34
0:[6]33
        2:[7]43
visit => [6]33; heap=>
        1:[7]36
0:[6]34
        2:[7]43
[6]34 already visited! heap=>
        1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

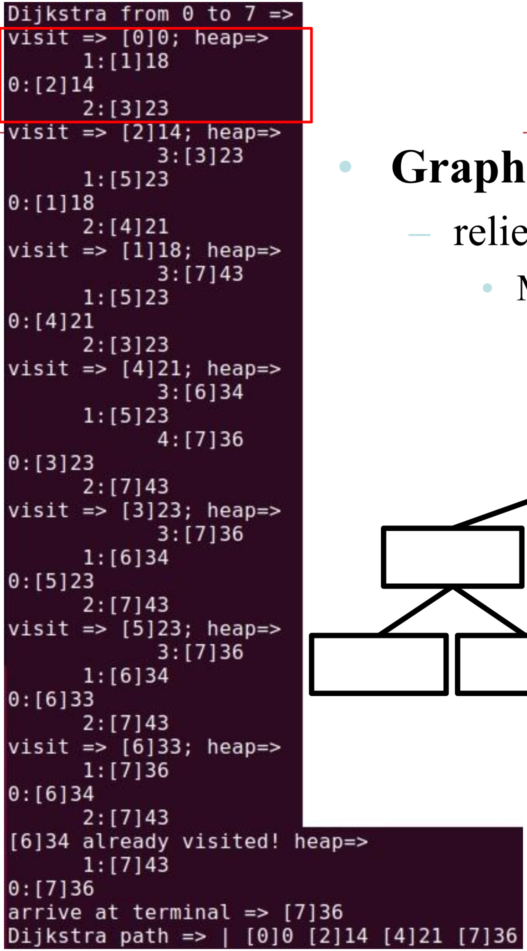
heap :



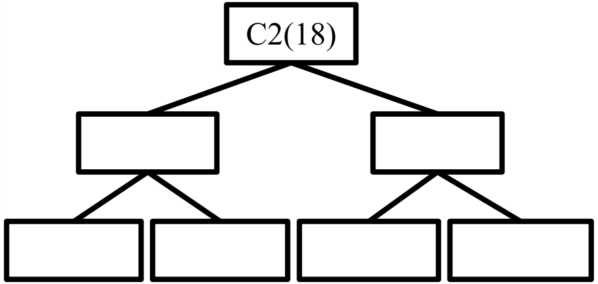


Graph

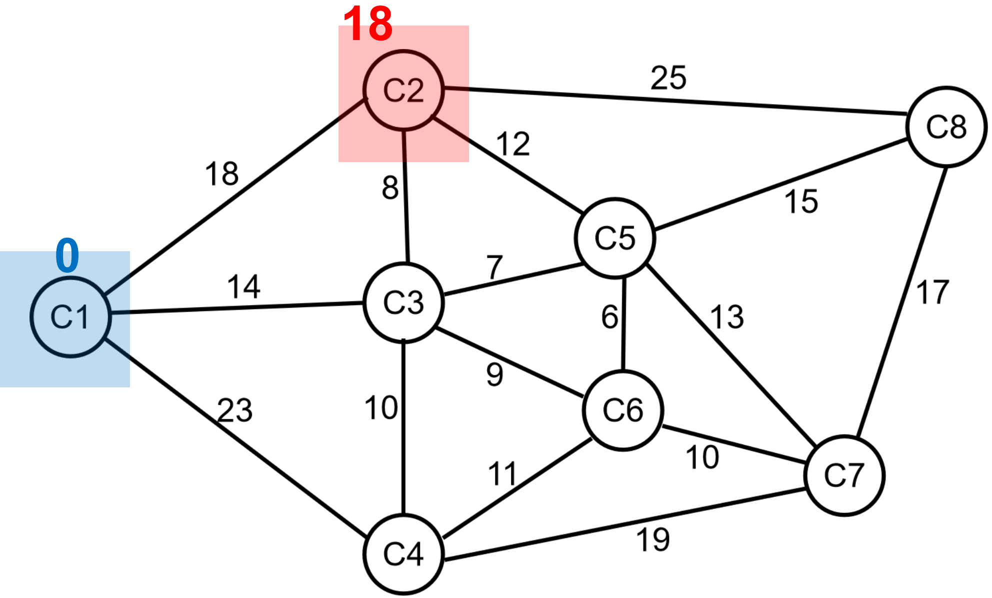
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

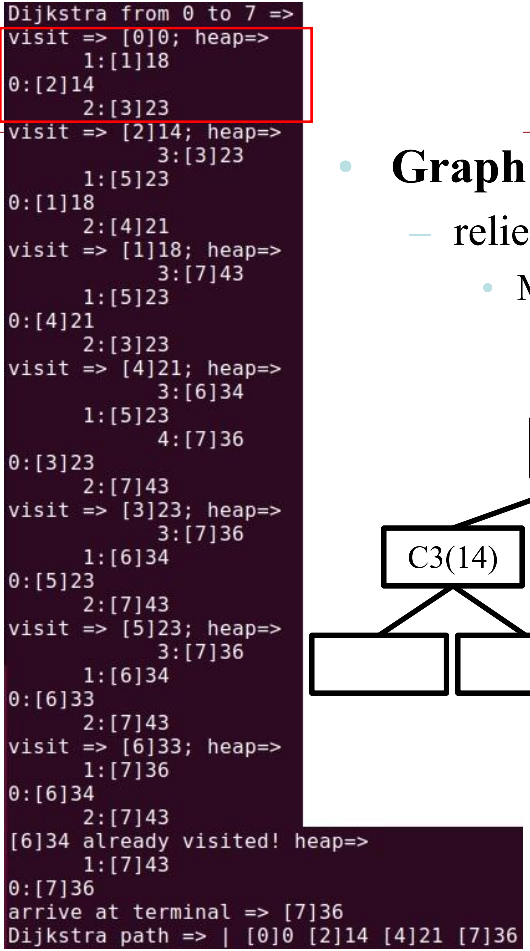


```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```



heap : C2



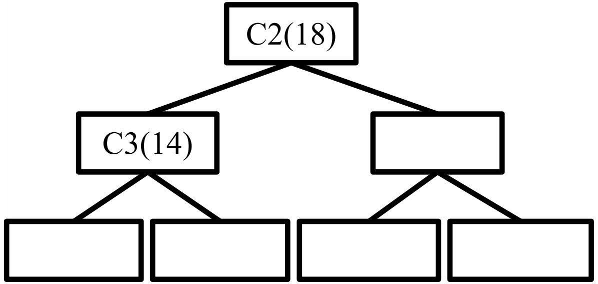


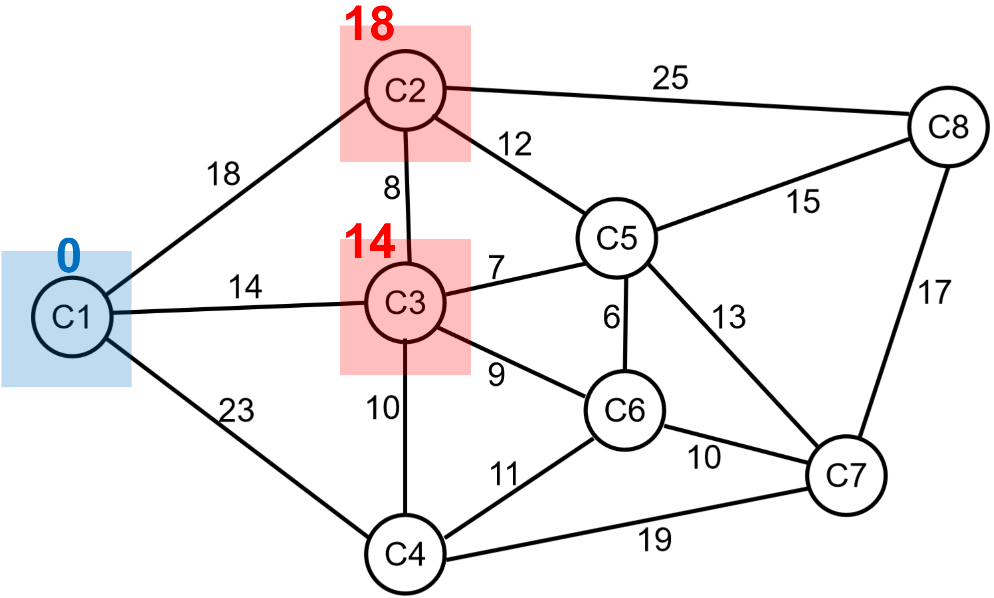
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

Graph

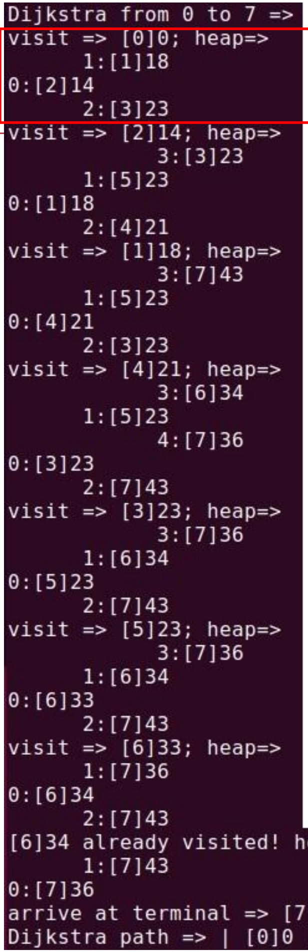
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap : C2 C3



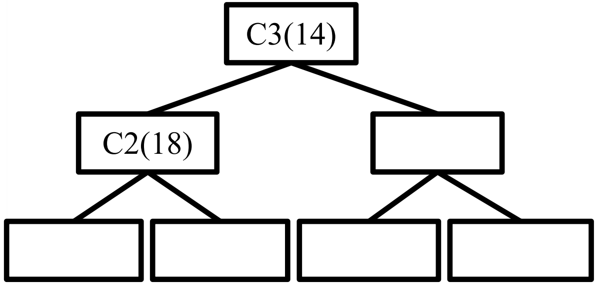


Graph

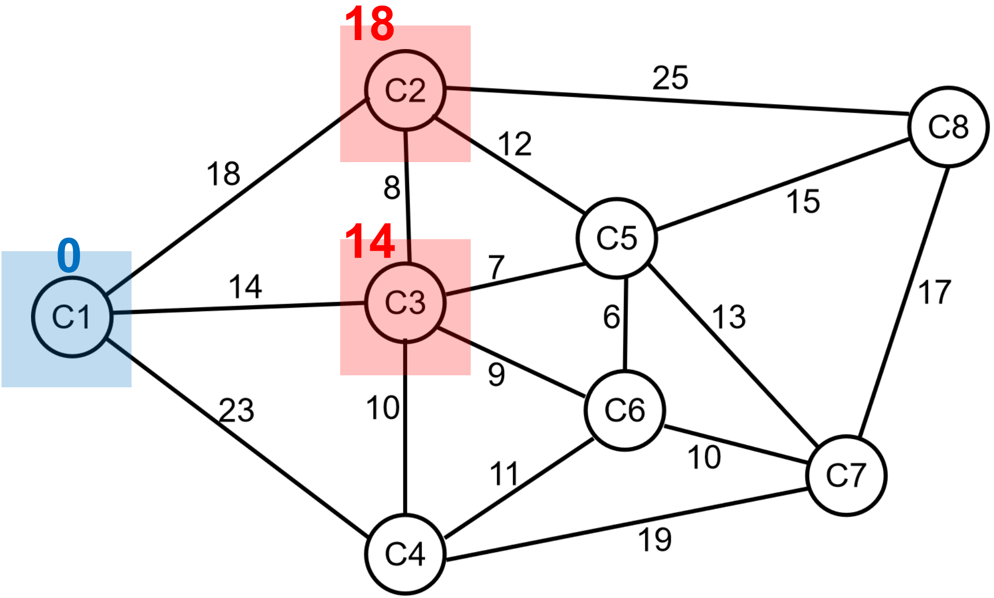


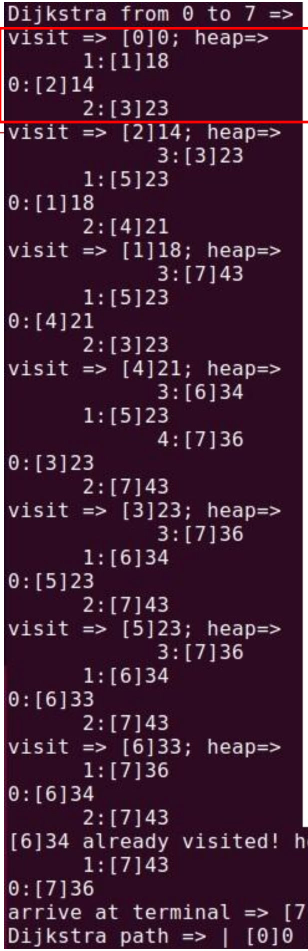
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



heap : C3 C2

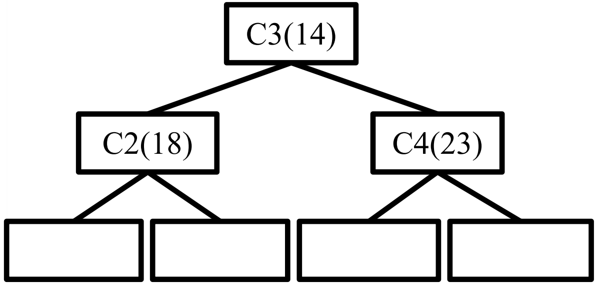




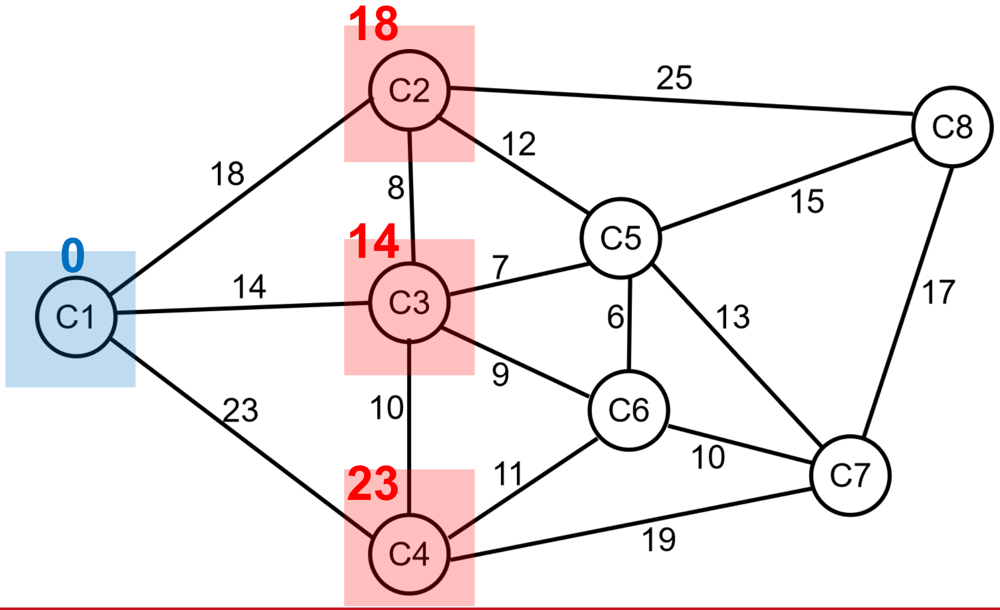
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

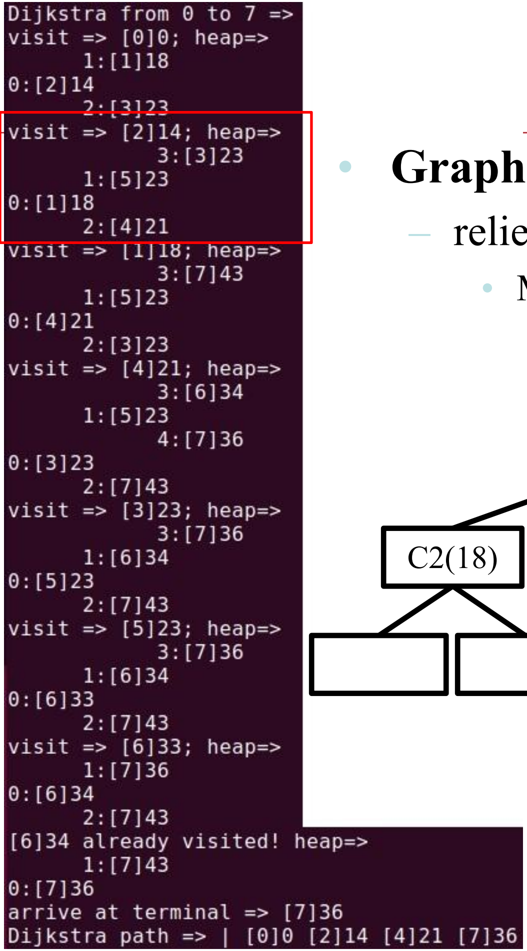


heap : C3 C2 C4

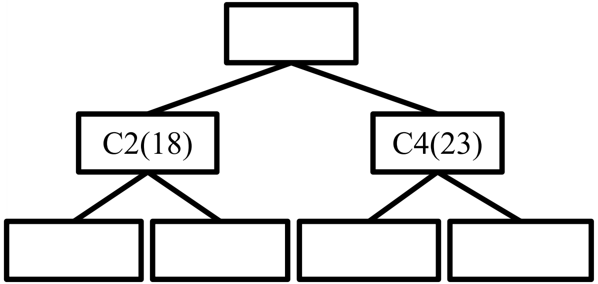


Graph

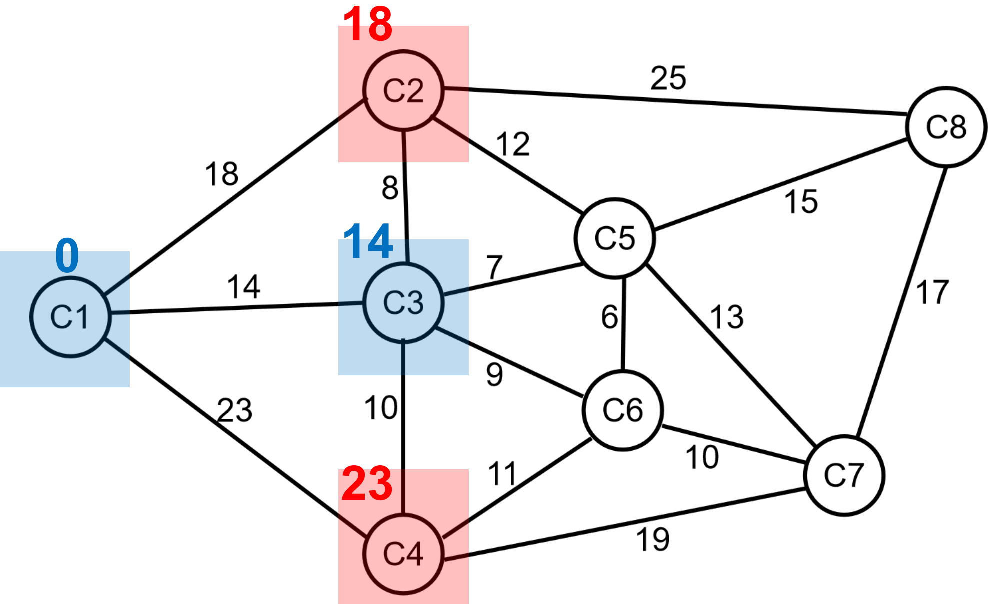
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



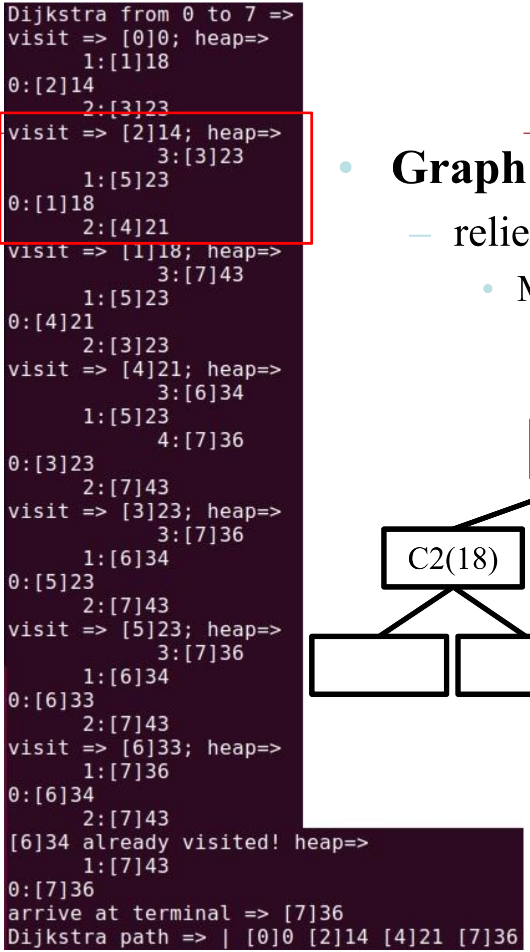
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```



heap : C2 C4



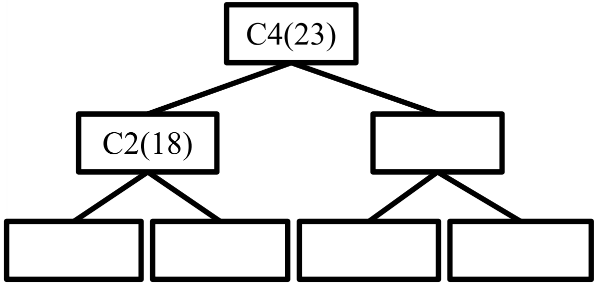
Graph

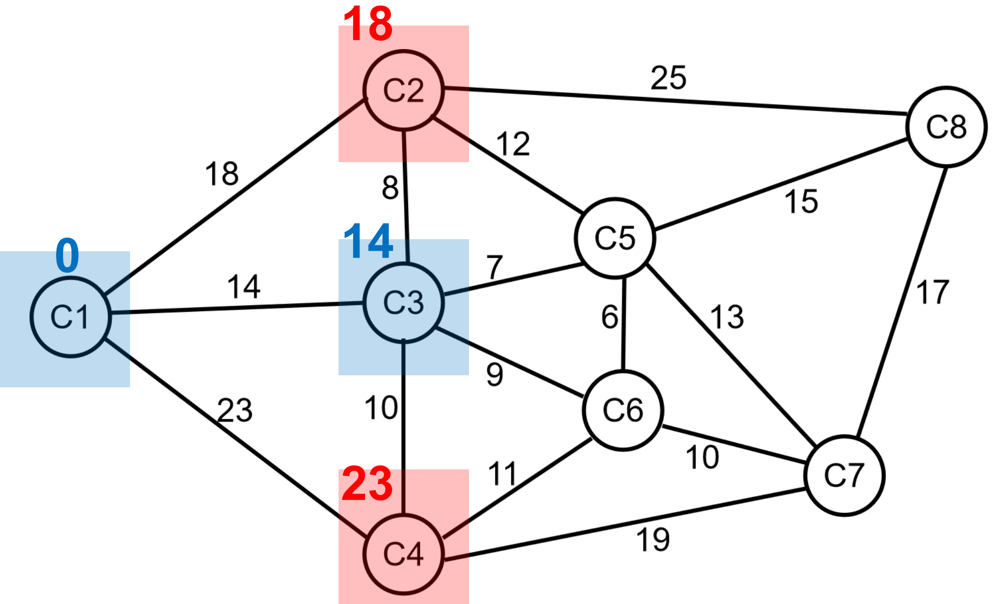


```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

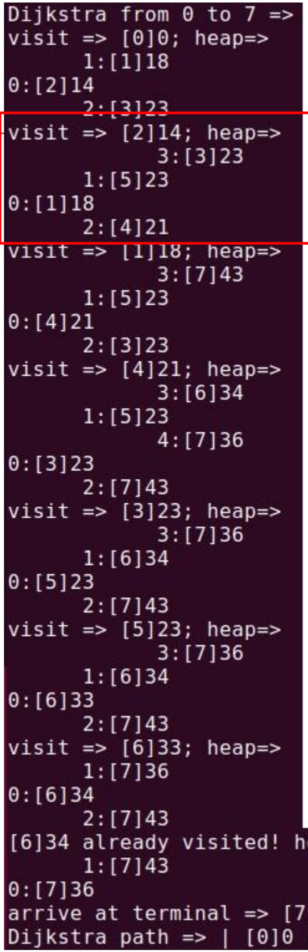
heap : C4 C2



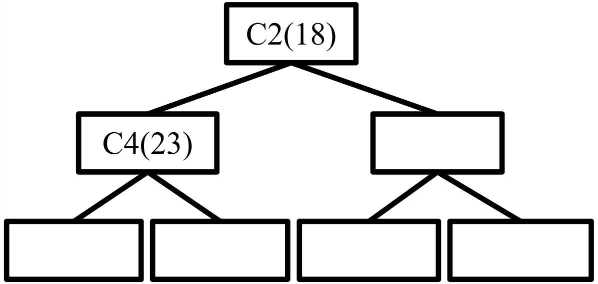


Graph

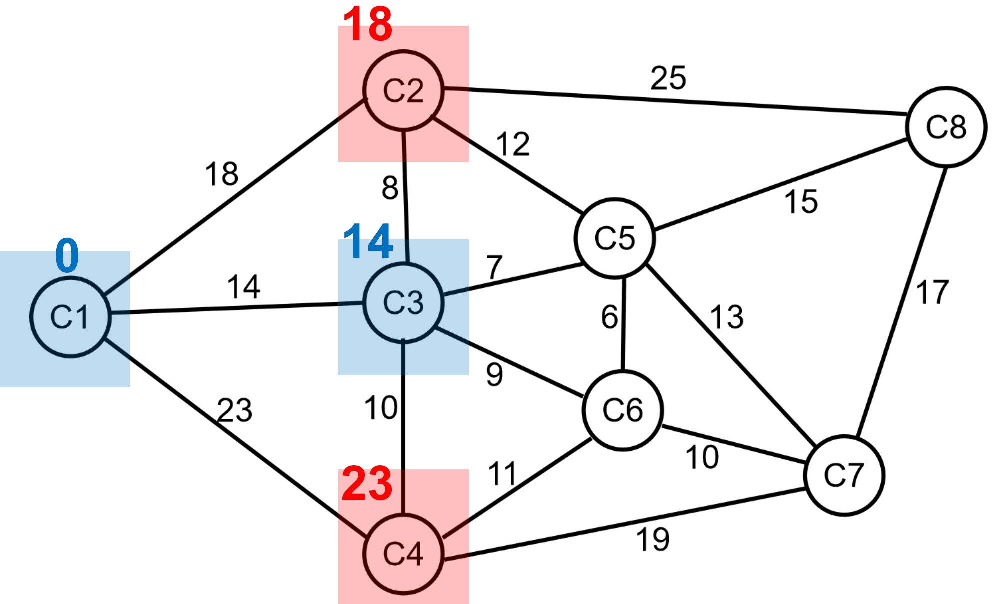
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



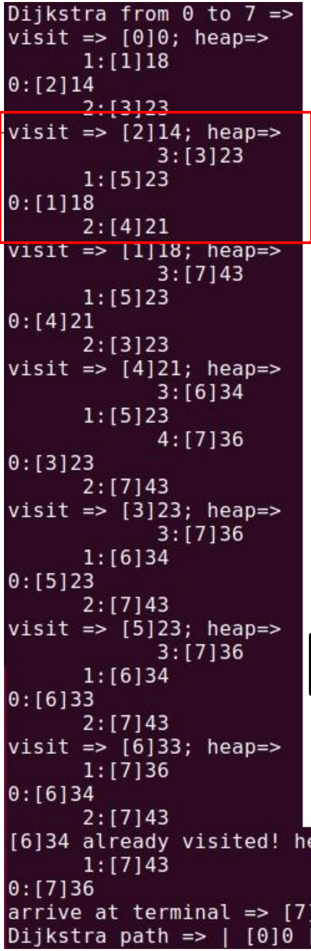
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```



heap : C2 C4



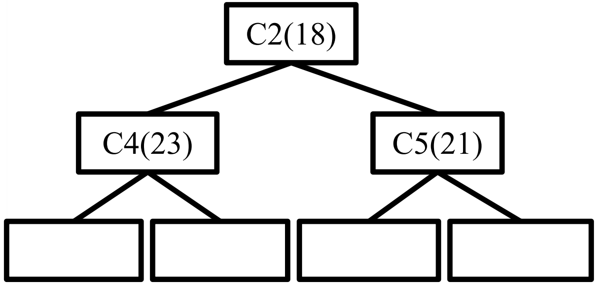
Graph

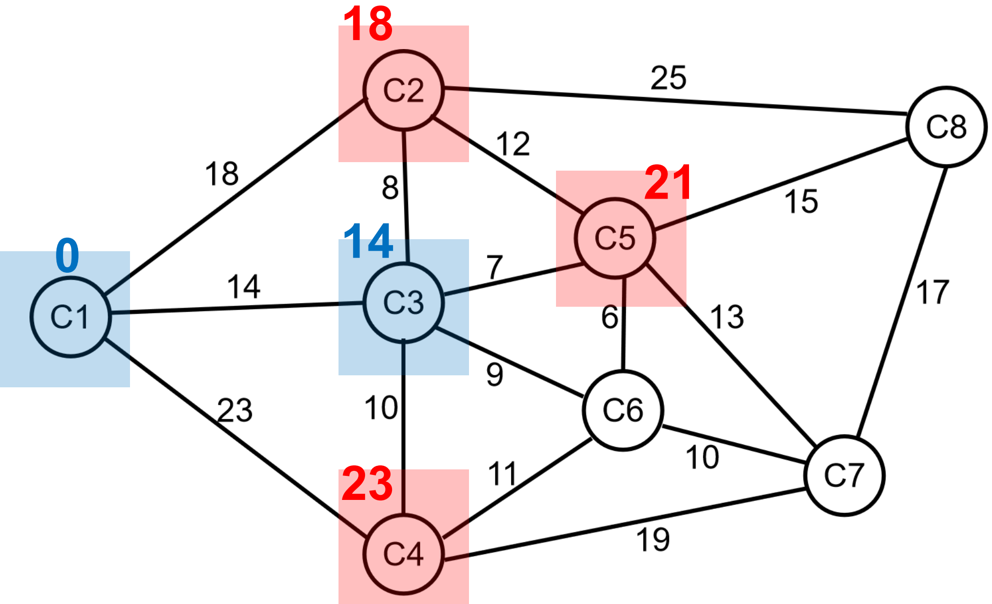


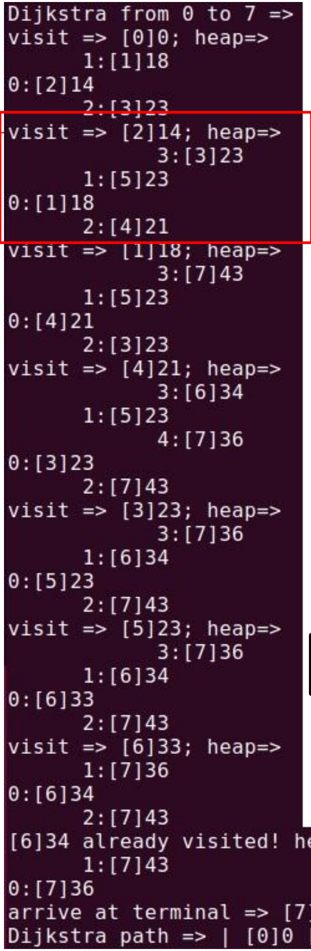
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap : C2 C4 C5





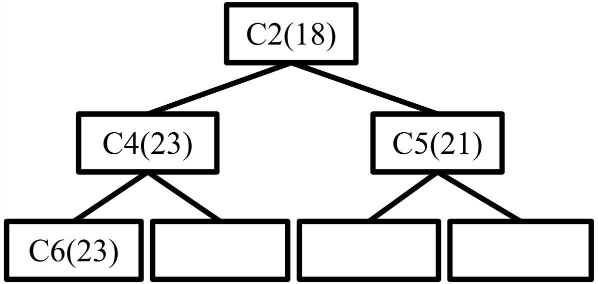


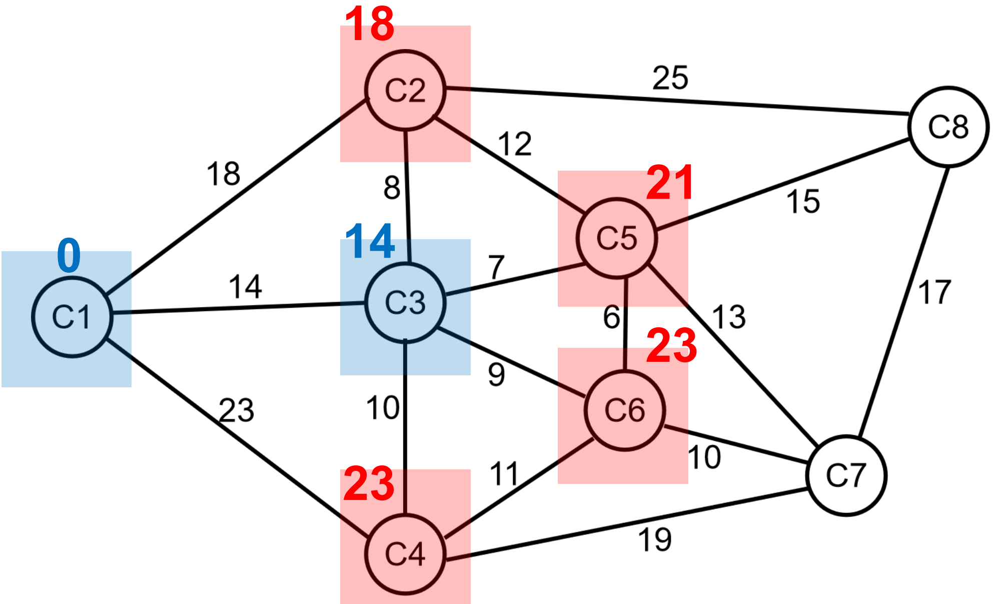
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

Graph

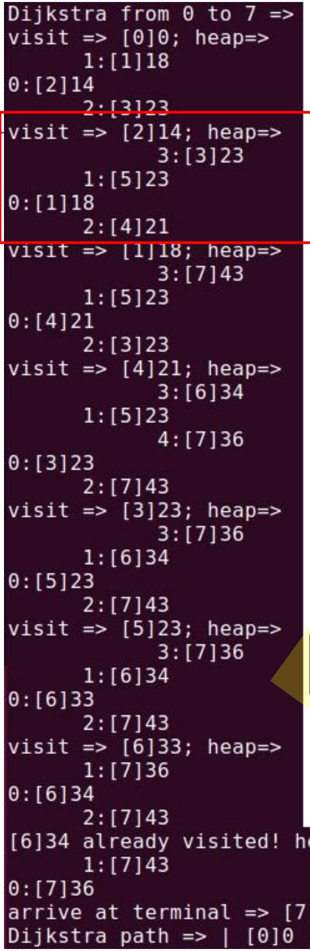
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap : C2 C4 C5 C6





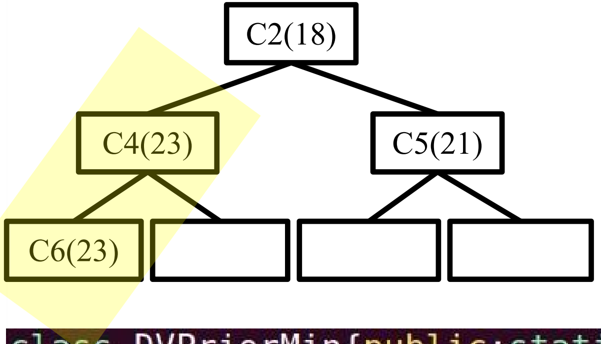
Graph

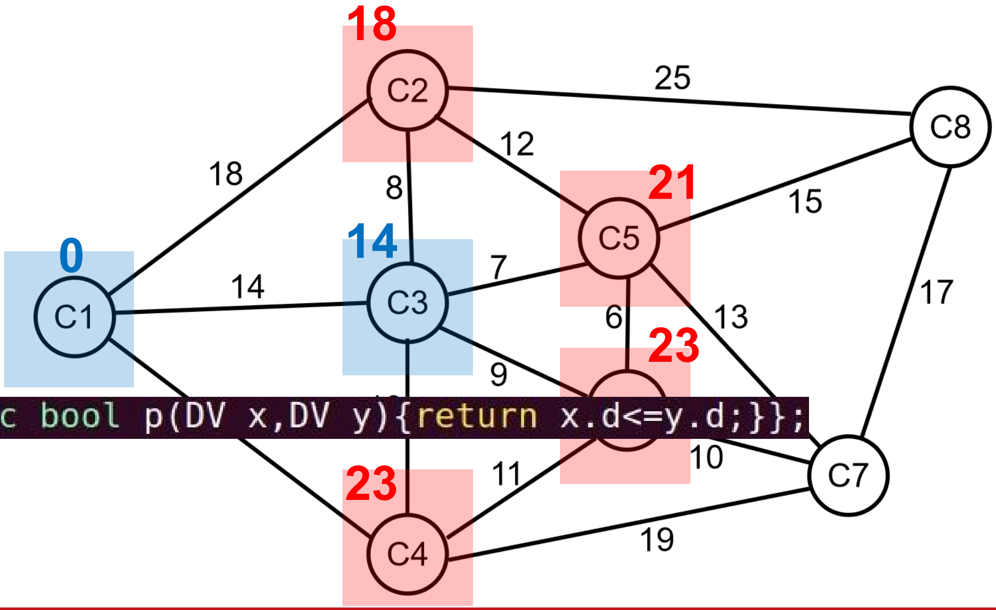


```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
        1:[1]18
0:[2]14
        2:[3]23
visit => [2]14; heap=>
                3:[3]23
        1:[5]23
0:[1]18
        2:[4]21
visit => [1]18; heap=>
                3:[7]43
        1:[5]23
0:[4]21
        2:[3]23
visit => [4]21; heap=>
                3:[6]34
        1:[5]23
                4:[7]36
0:[3]23
        2:[7]43
visit => [3]23; heap=>
                3:[7]36
        1:[6]34
0:[5]23
        2:[7]43
visit => [5]23; heap=>
                3:[7]36
        1:[6]34
0:[6]33
        2:[7]43
visit => [6]33; heap=>
        1:[7]36
0:[6]34
        2:[7]43
[6]34 already visited! heap=>
        1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap : C2 C4 C5 C6

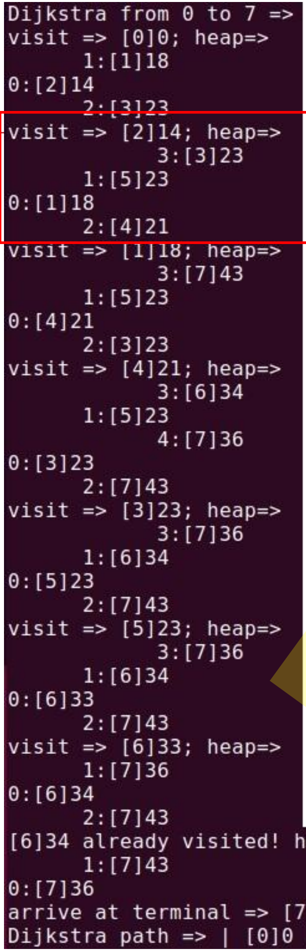




```
class DVPriorMin{public:static bool p(DV x,DV y){return x.d<=y.d;}};
```

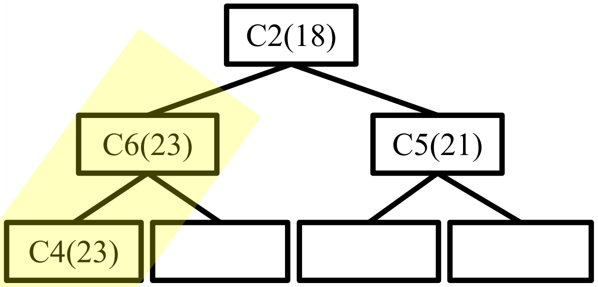
Graph

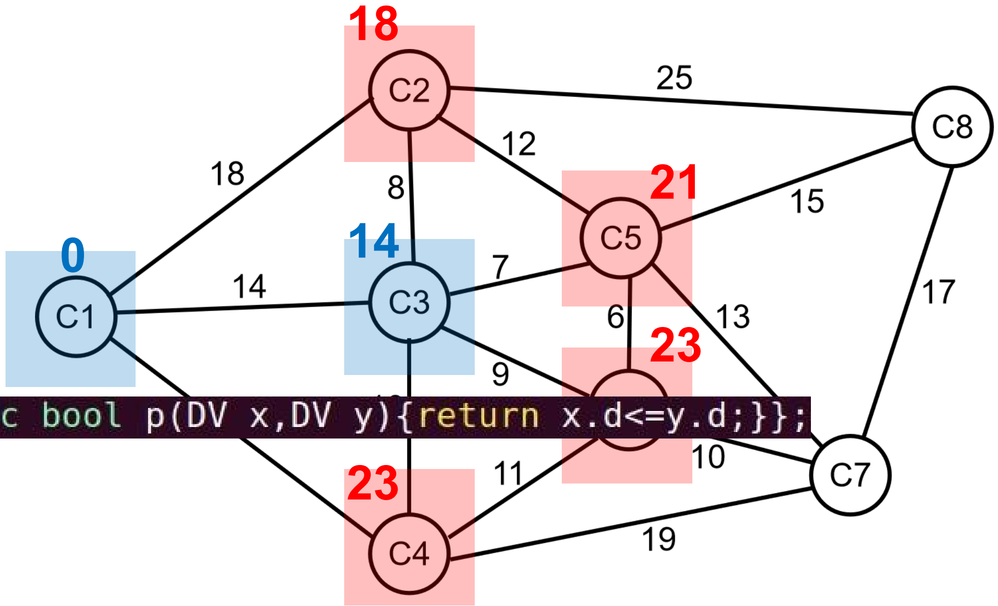
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

heap : C2 C6 C5 C4

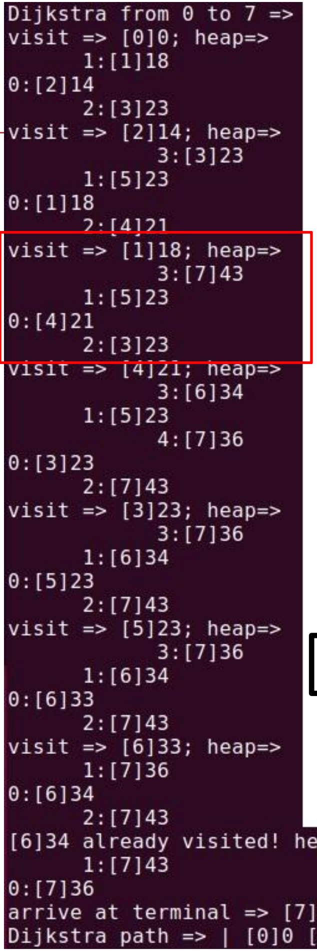




```
class DVPriorMin{public:static bool p(DV x,DV y){return x.d<=y.d;}};
```

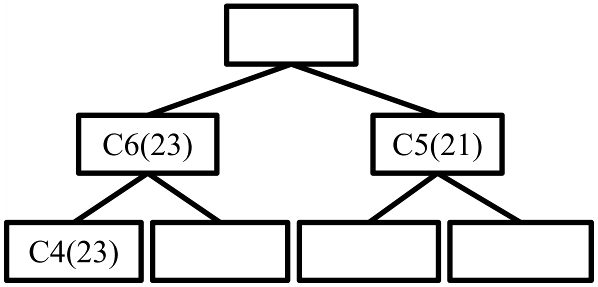
Graph

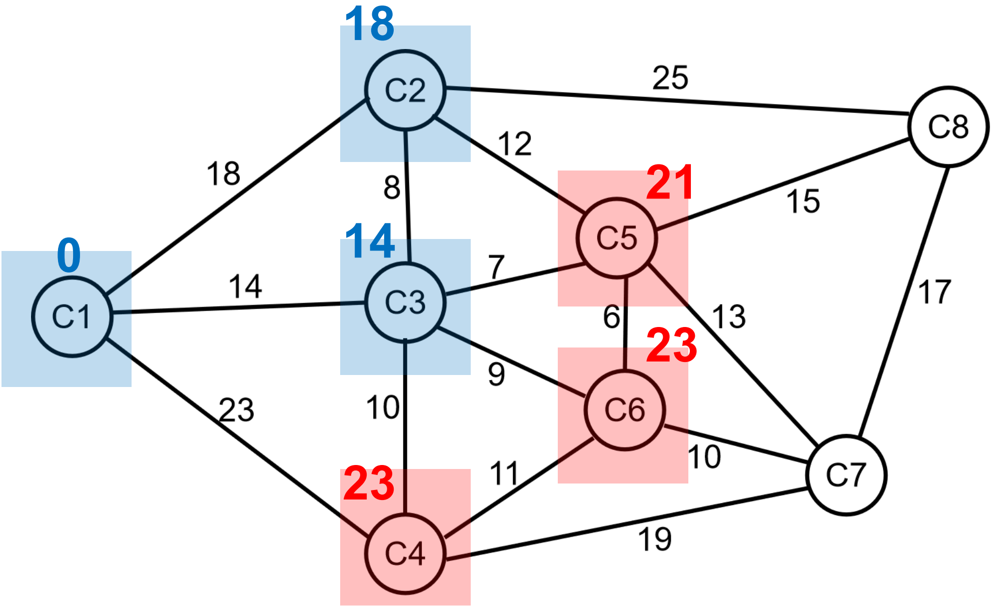
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

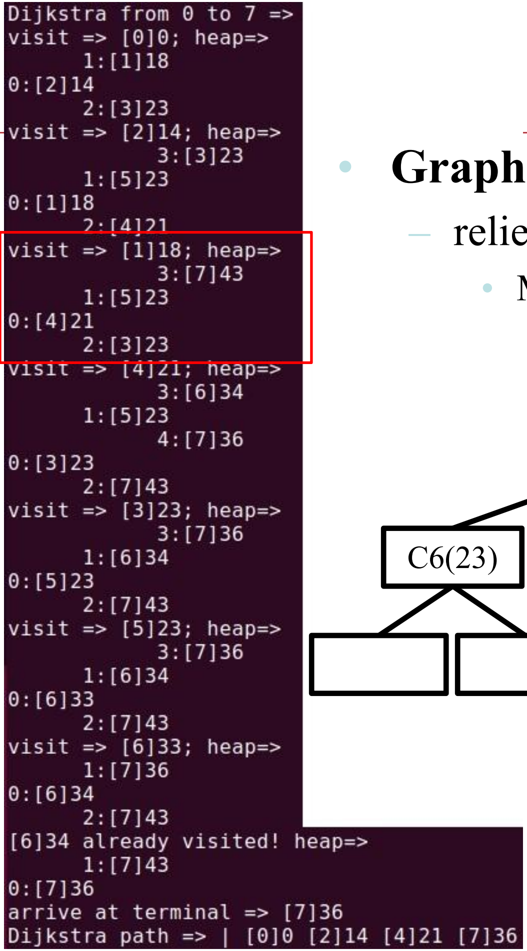
heap :   C6 C5 C4





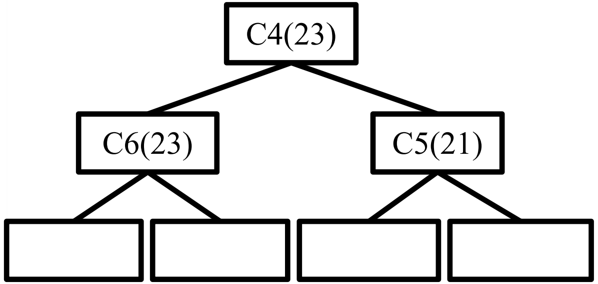
Graph

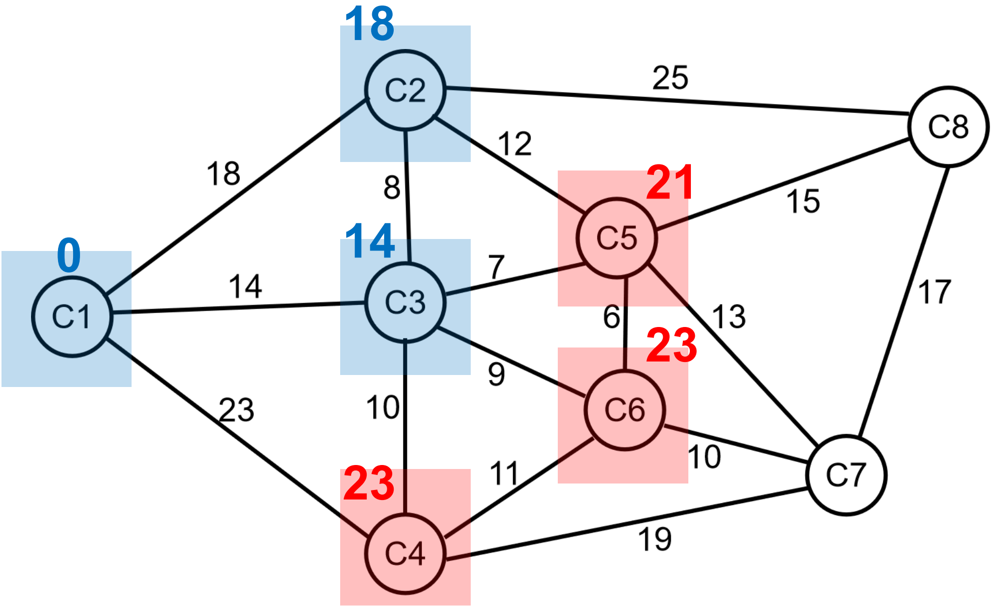
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

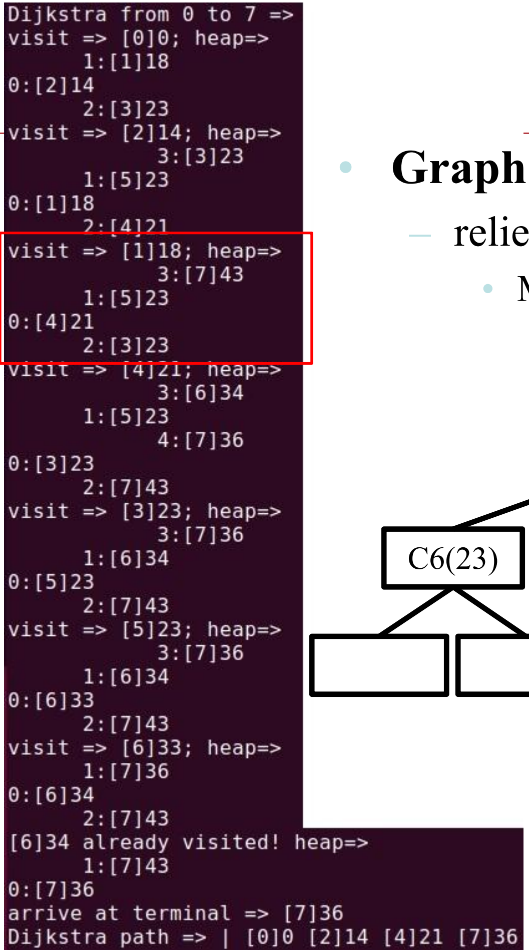
heap : C4 C6 C5





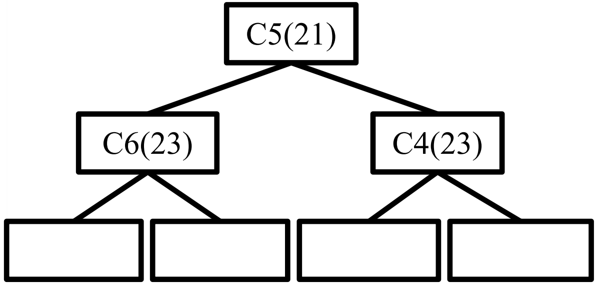
Graph

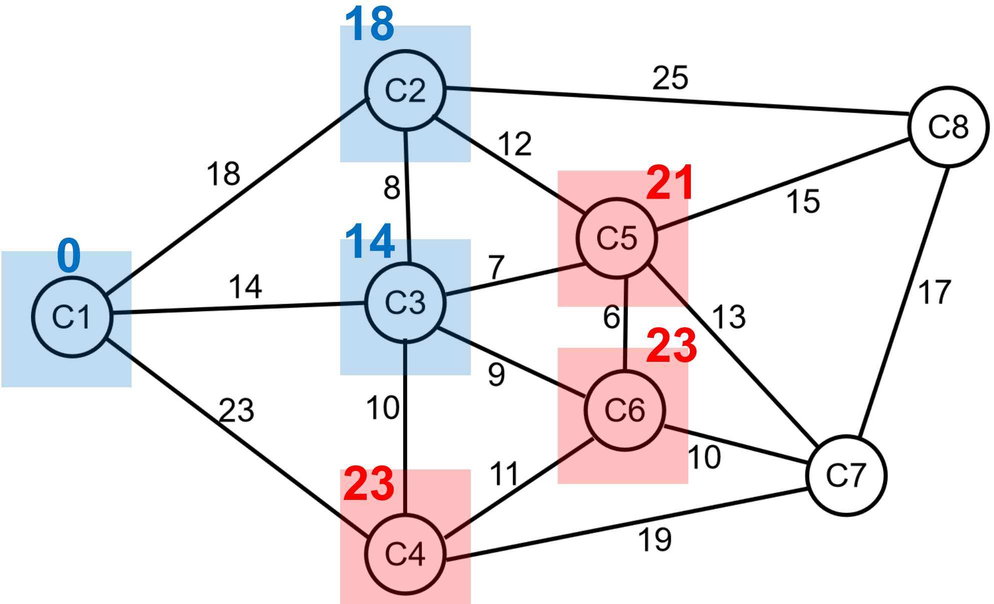
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

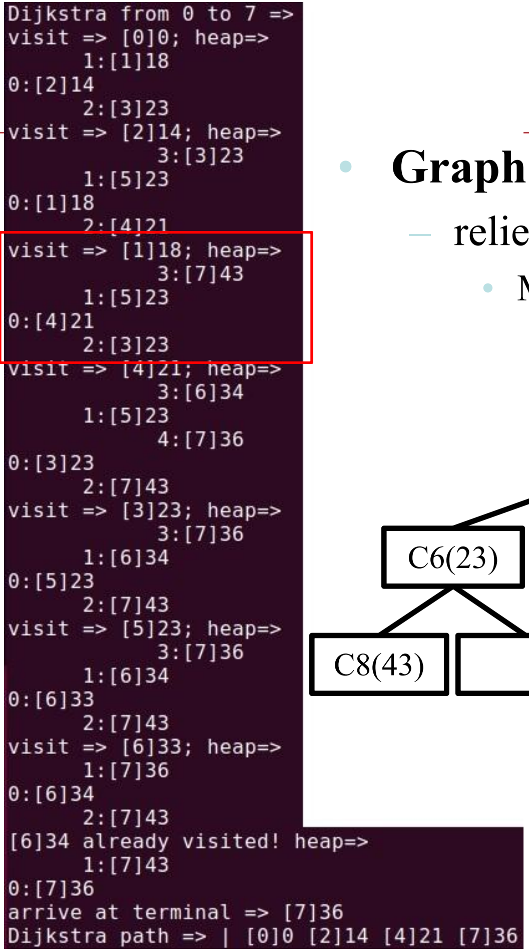
heap : C5 C6 C4





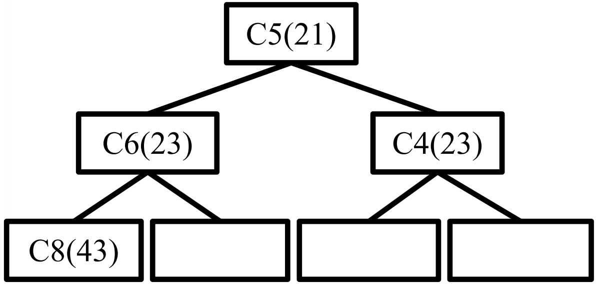
Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

heap : C5 C6 C4 C8

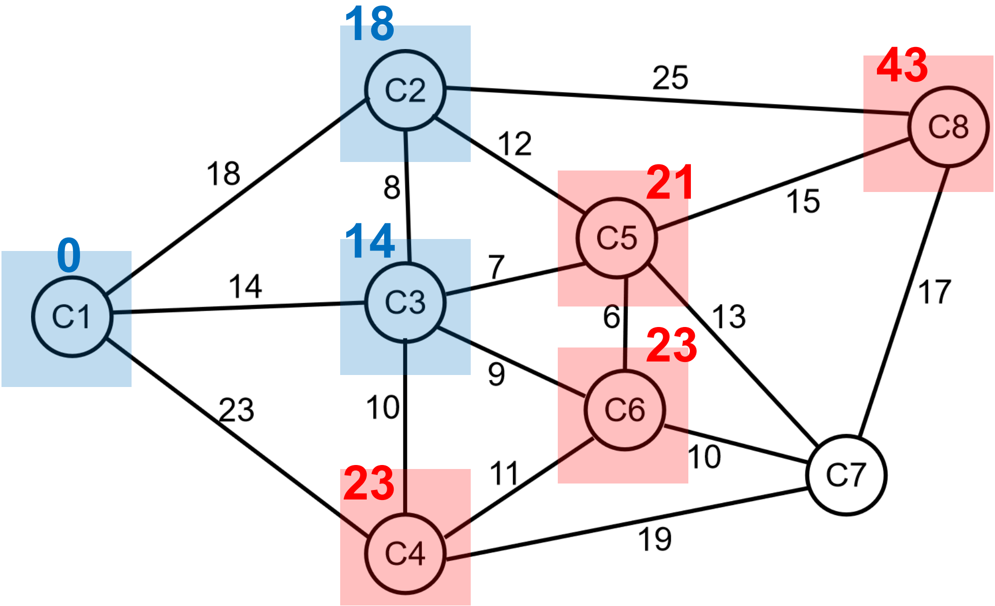


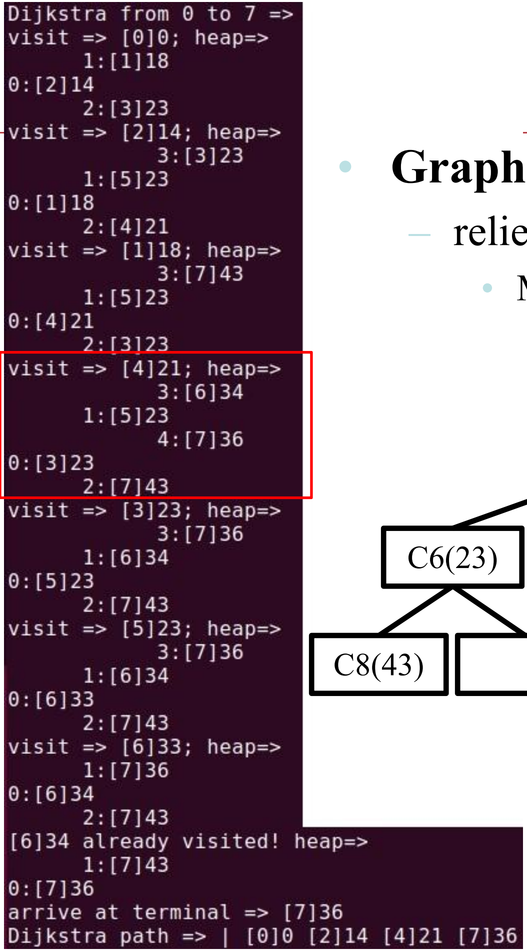
C5(21)

C6(23)

C4(23)

C8(43)



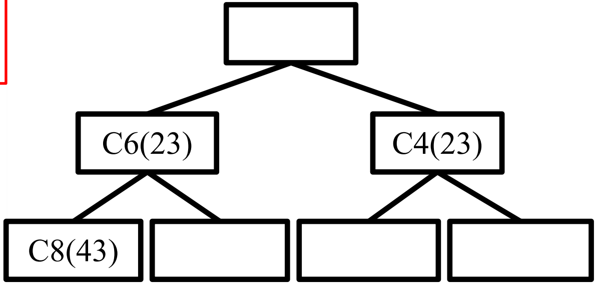


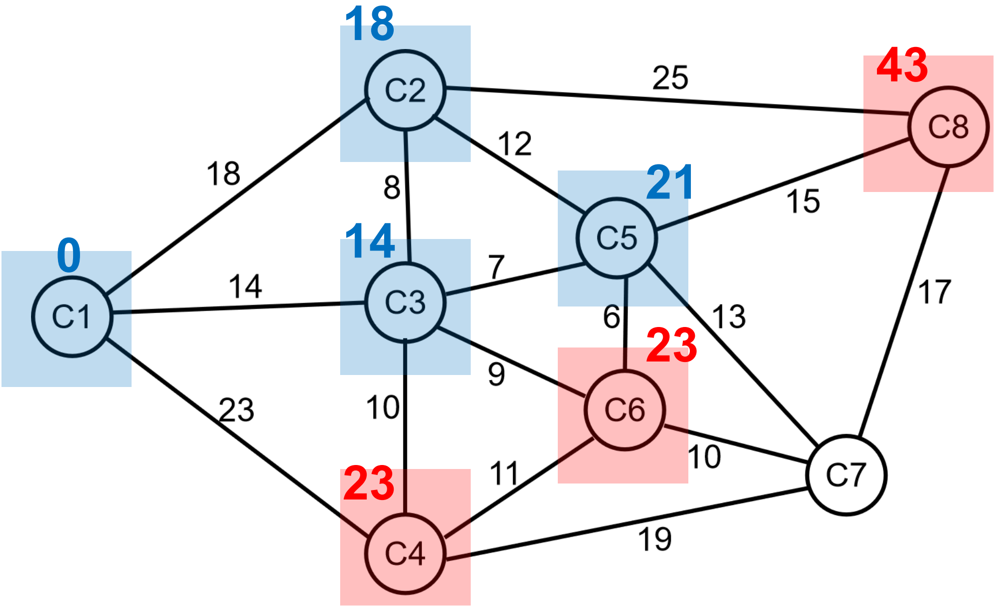
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

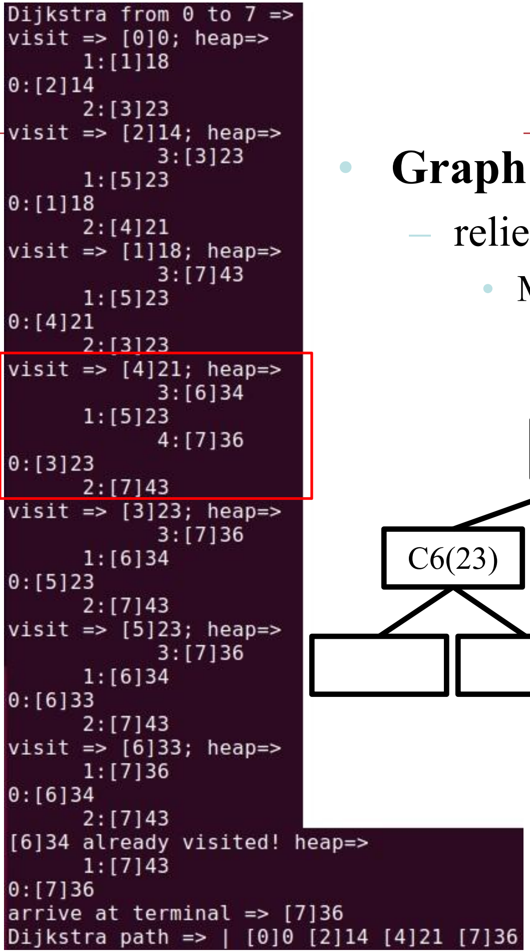
Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap :   C6 C4 C8





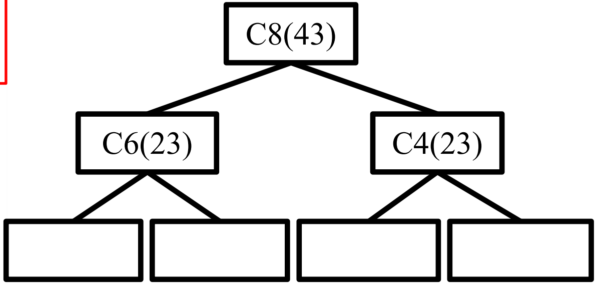


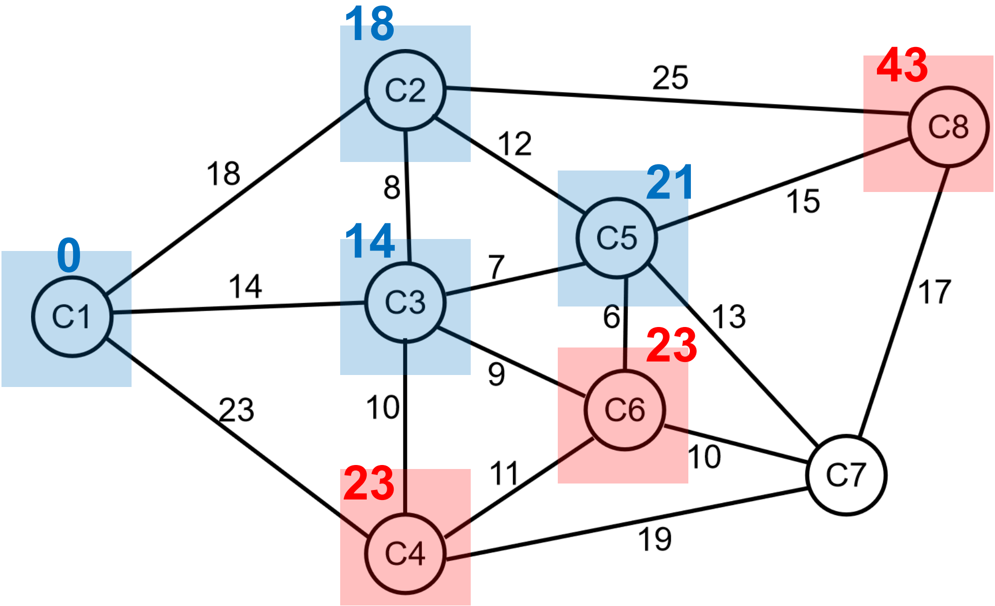
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

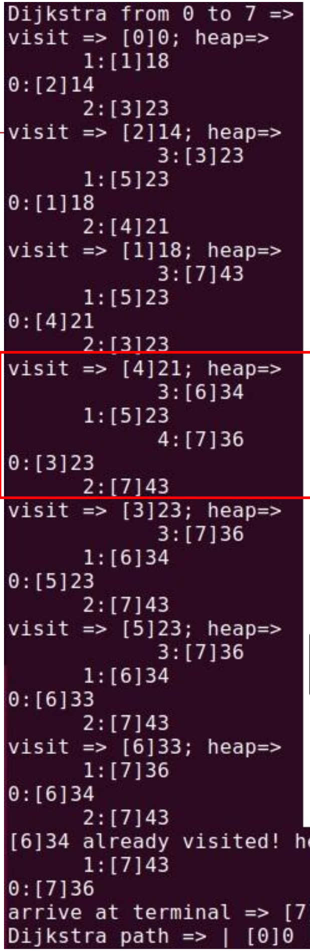
Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap : C8 C6 C4



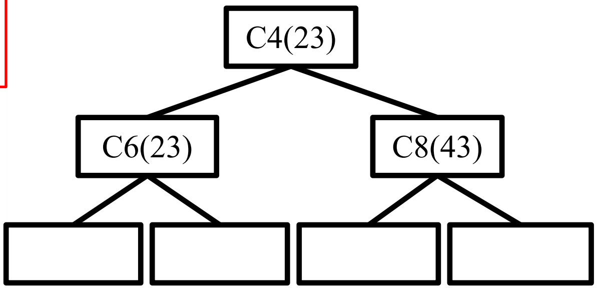




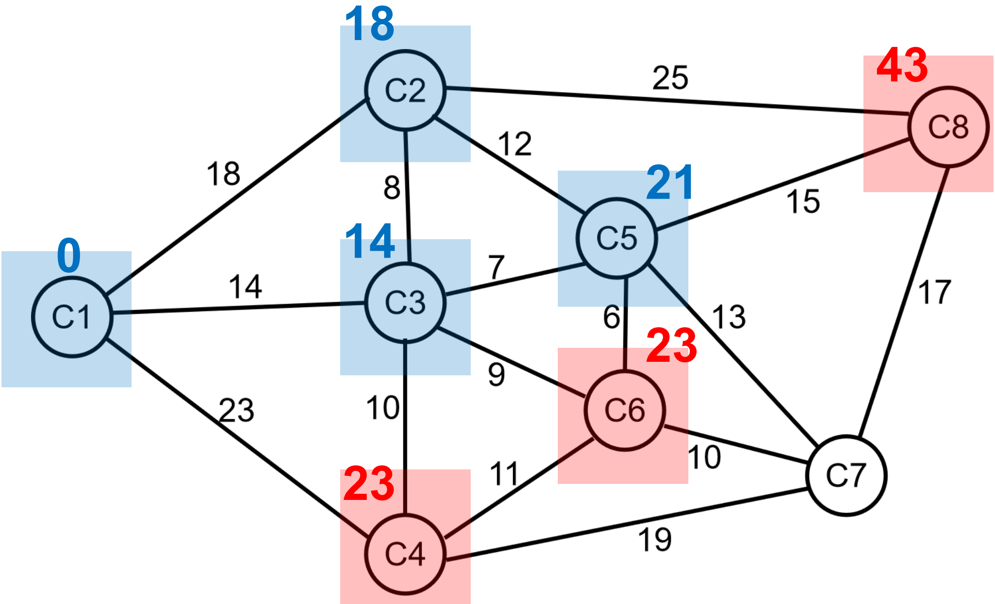
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

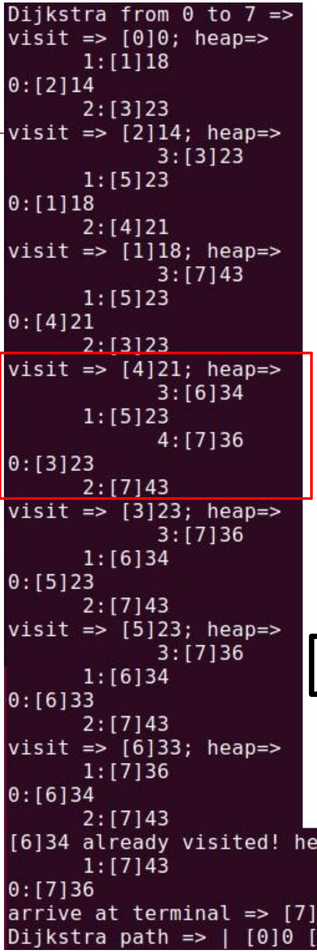


heap : C4 C6 C8

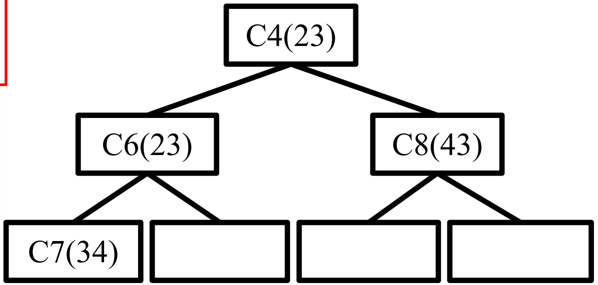


Graph

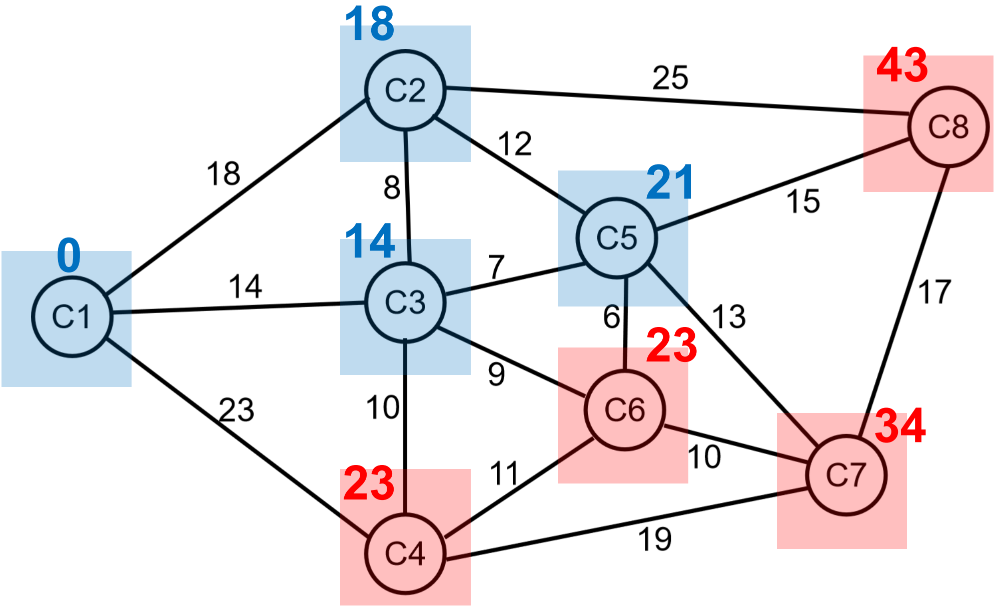
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



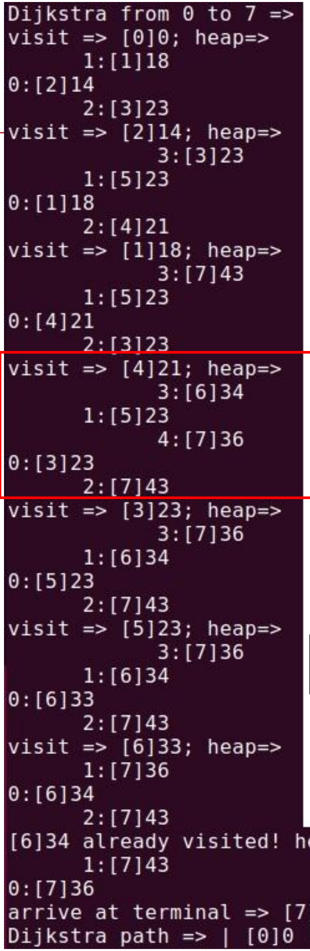
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```



heap : C4 C6 C8 C7



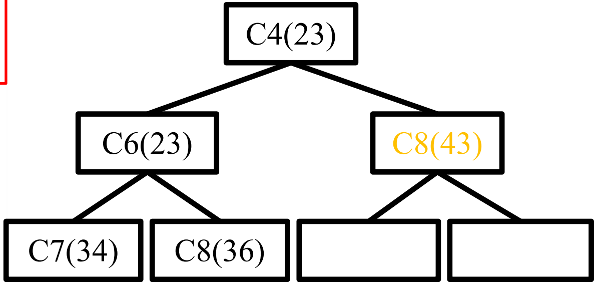
Graph



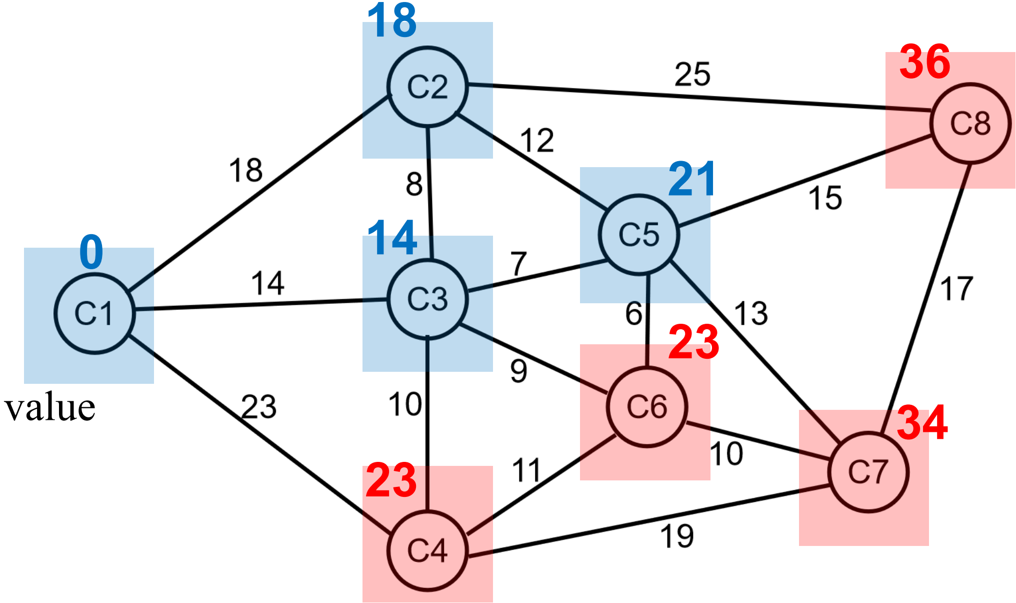
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

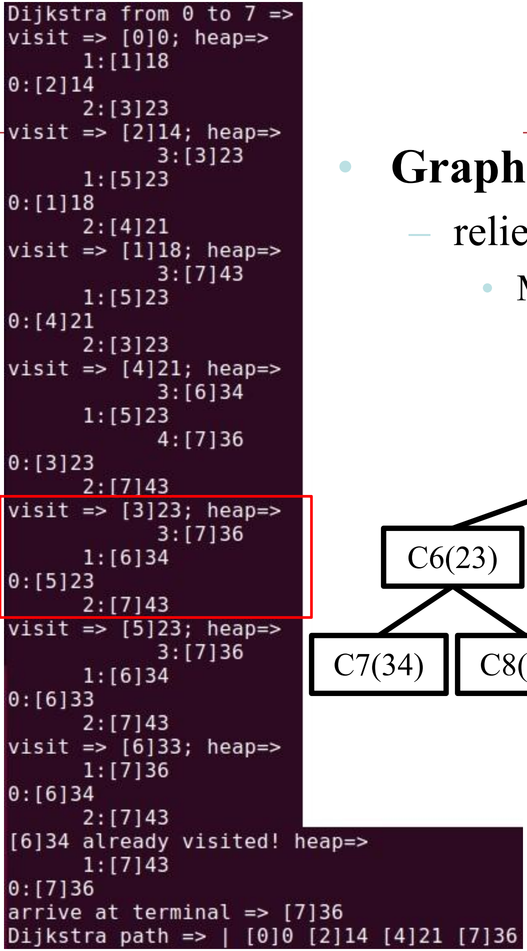
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap : C4 C6 C8 C7 C8



just insert the vertex with updated value as a new node into the heap



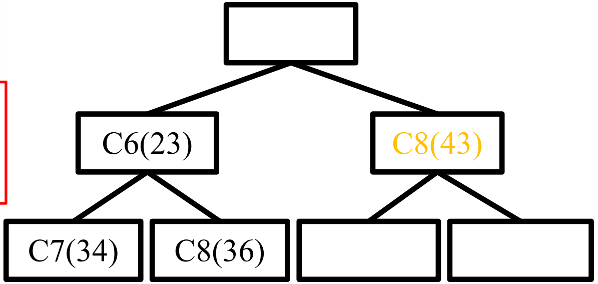


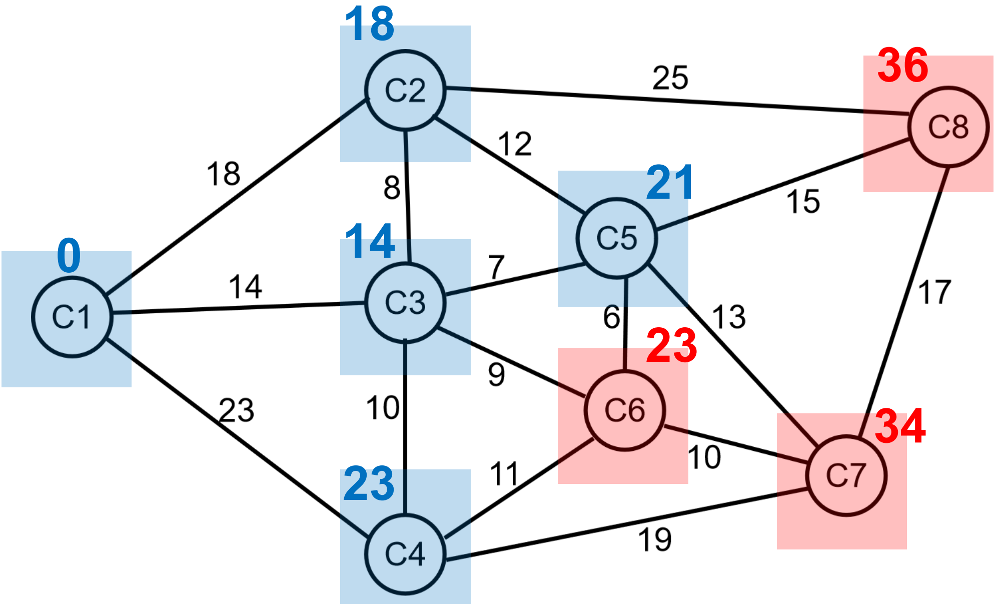
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

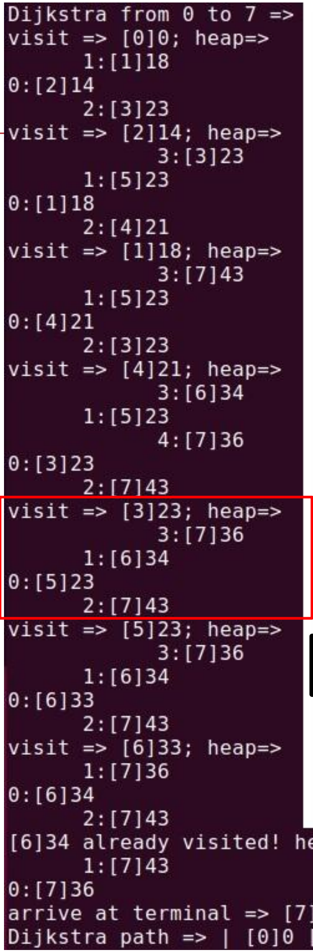
heap : C6 C8 C7 C8





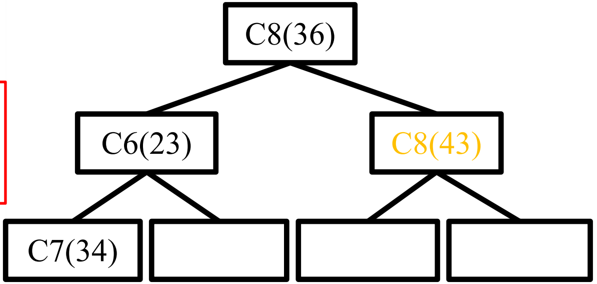
Graph

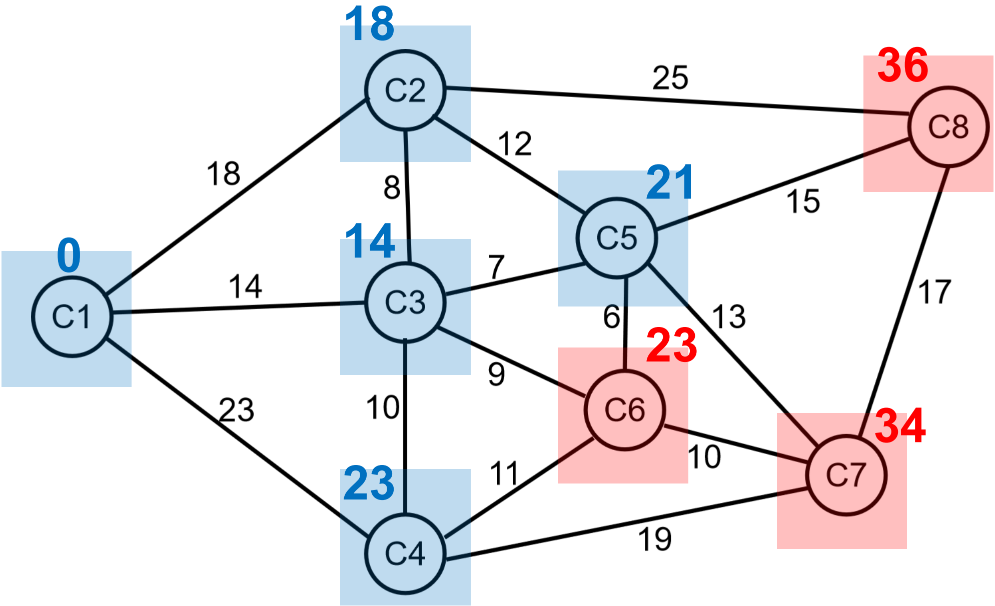
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

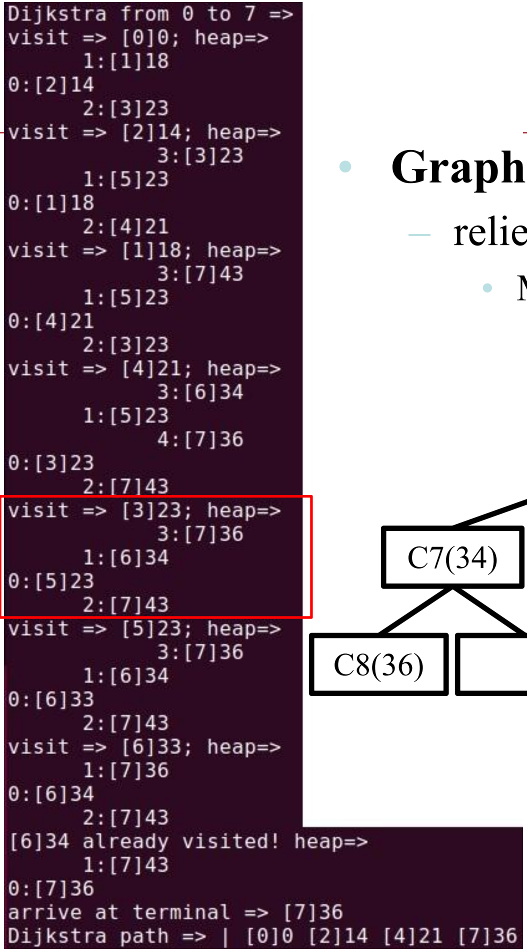


```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

heap : C8 C6 C8 C7



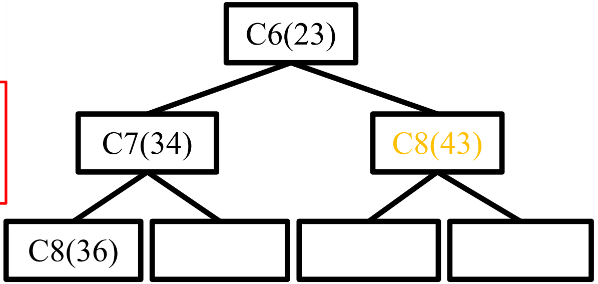




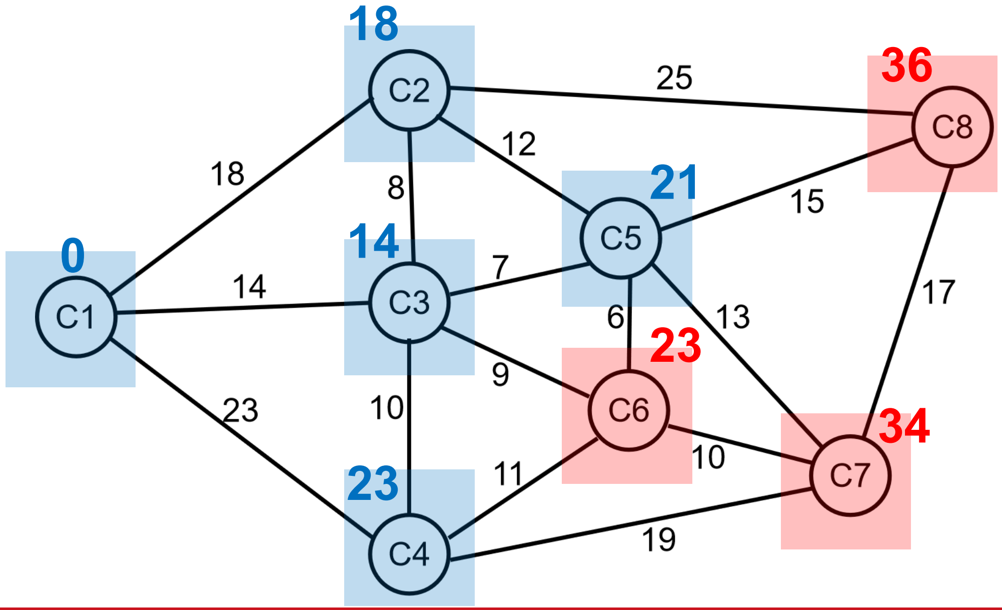
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

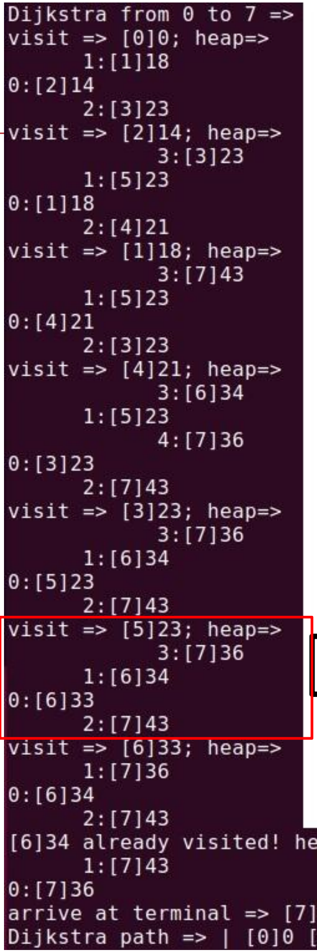


heap : C6 C7 C8 C8

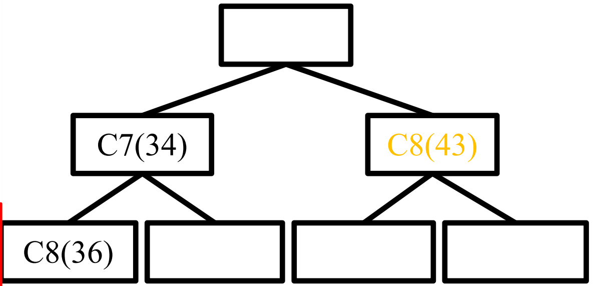


Graph

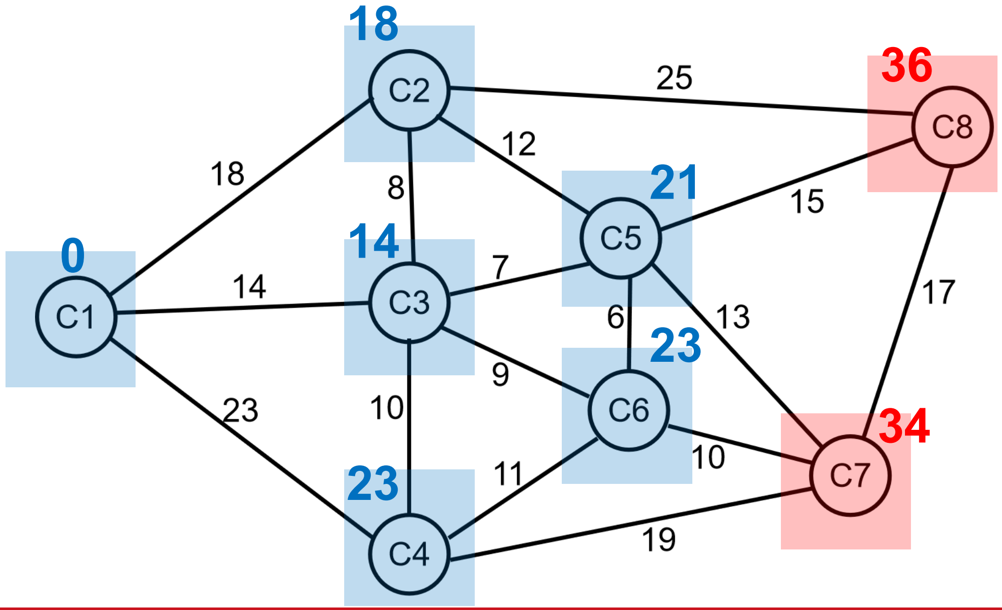
- **Graph - *Dijkstra* algorithm -** single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
        1:[1]18
0:[2]14
        2:[3]23
visit => [2]14; heap=>
                3:[3]23
        1:[5]23
0:[1]18
        2:[4]21
visit => [1]18; heap=>
                3:[7]43
        1:[5]23
0:[4]21
        2:[3]23
visit => [4]21; heap=>
                3:[6]34
        1:[5]23
                4:[7]36
0:[3]23
        2:[7]43
visit => [3]23; heap=>
                3:[7]36
        1:[6]34
0:[5]23
        2:[7]43
visit => [5]23; heap=>
                3:[7]36
        1:[6]34
0:[6]33
        2:[7]43
visit => [6]33; heap=>
        1:[7]36
0:[6]34
        2:[7]43
[6]34 already visited! heap=>
        1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

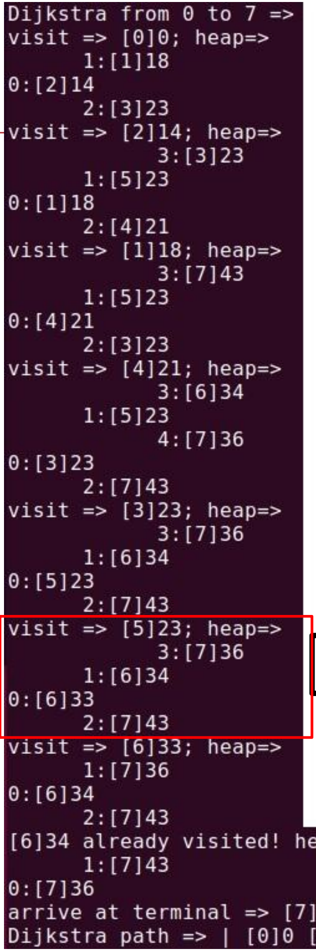


heap :   C7 C8 C8



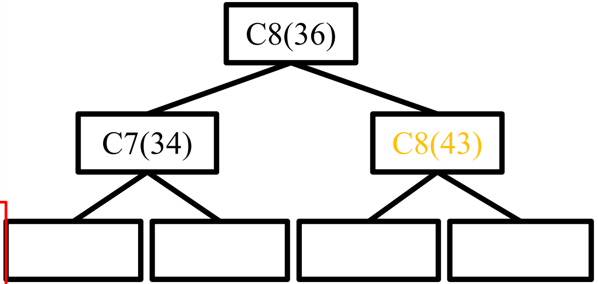
Graph

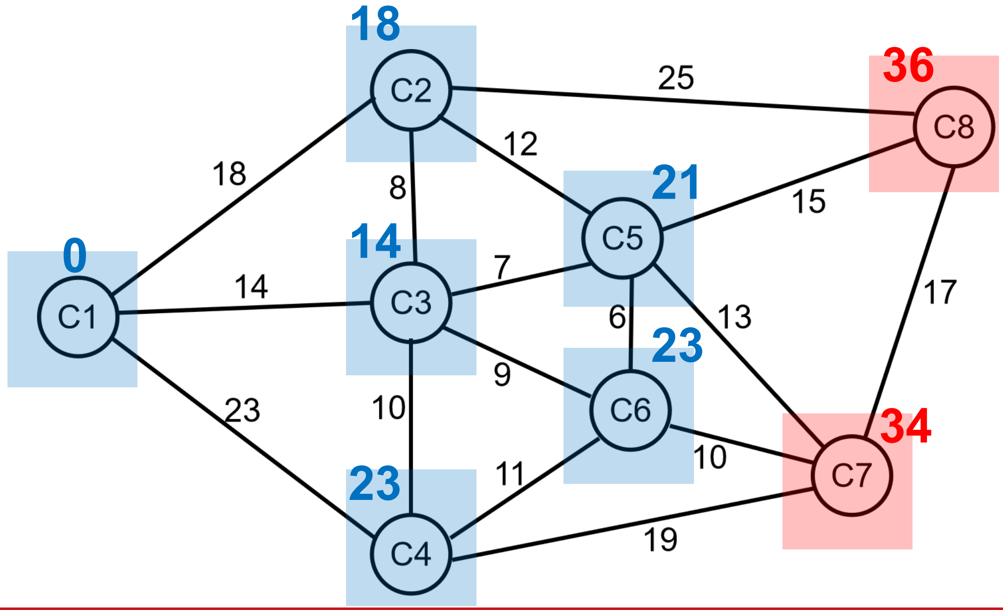
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

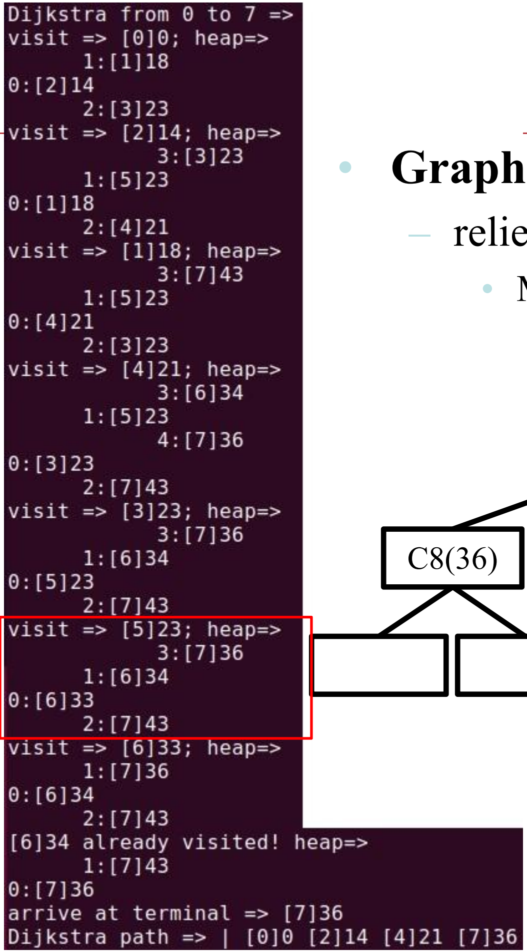
heap : C8 C7 C8





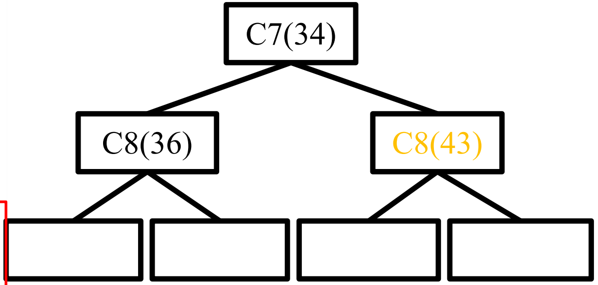
Graph

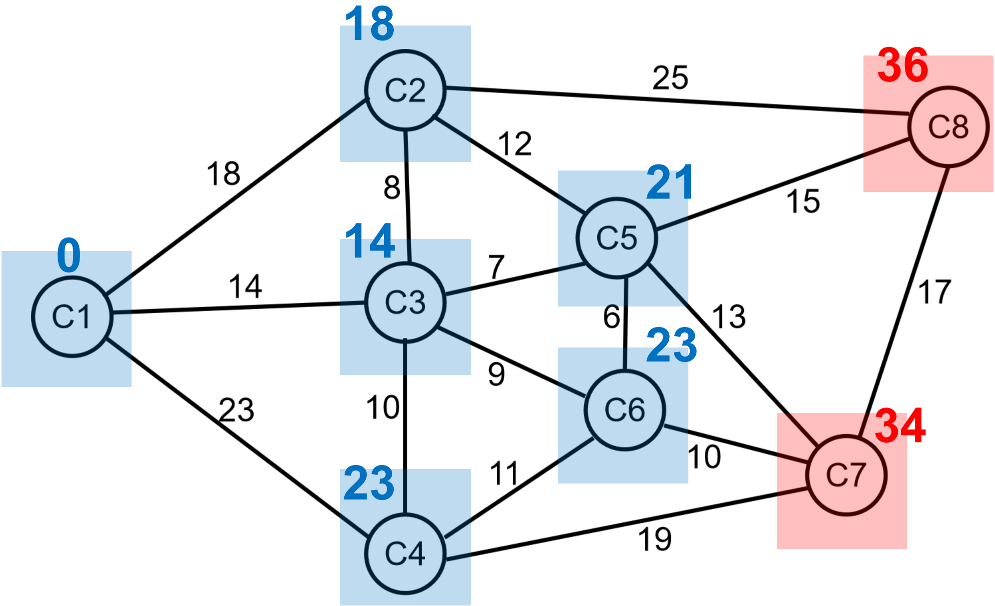
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

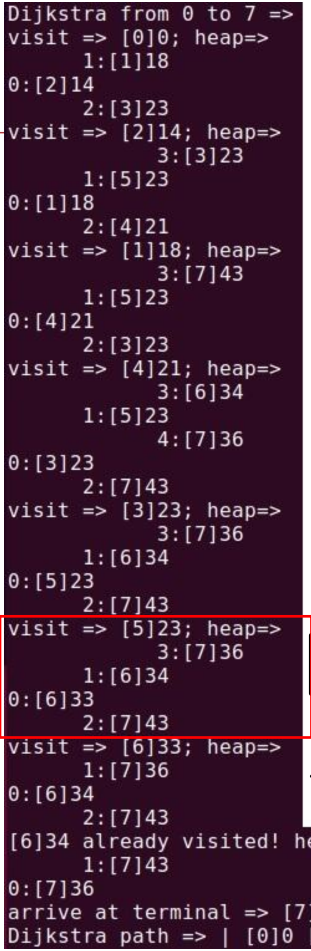
heap : C7 C8 C8



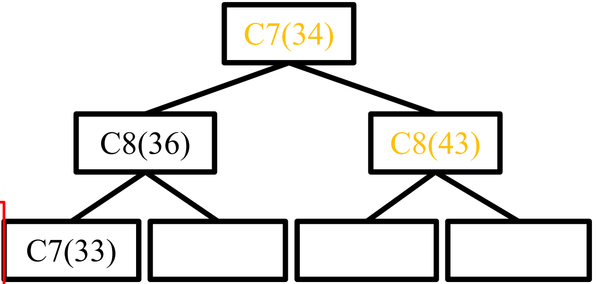


Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

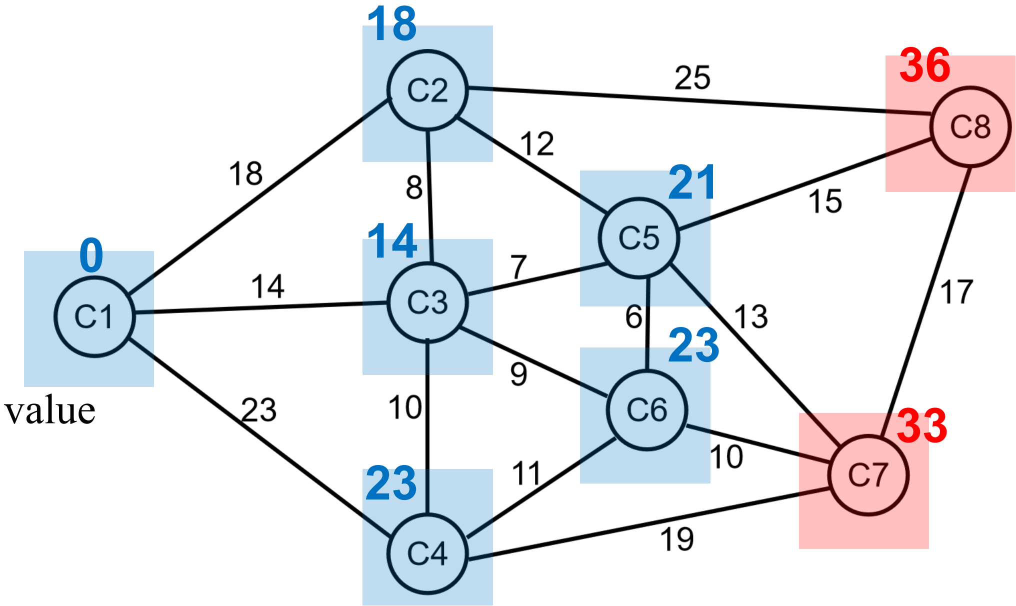


```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```



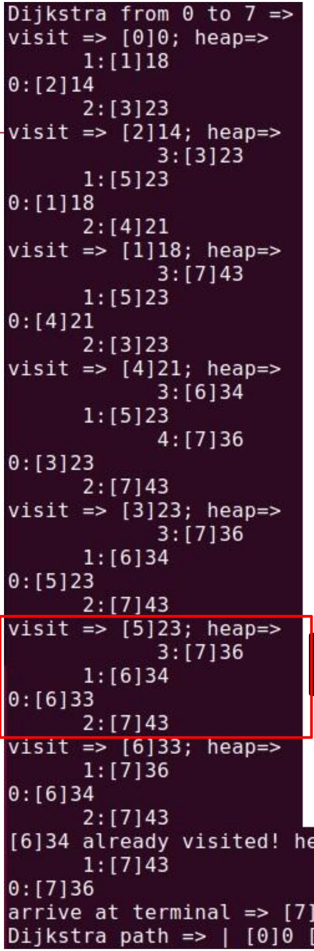
just insert the vertex with updated value as a new node into the heap

heap : C7 C8 C8 C7



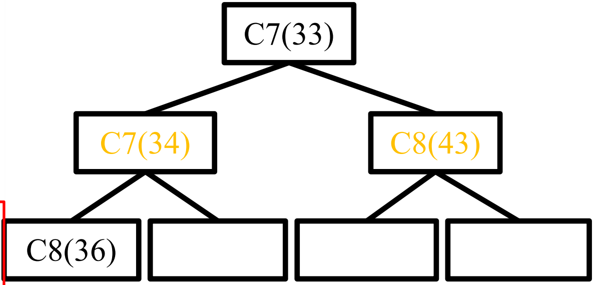
Graph

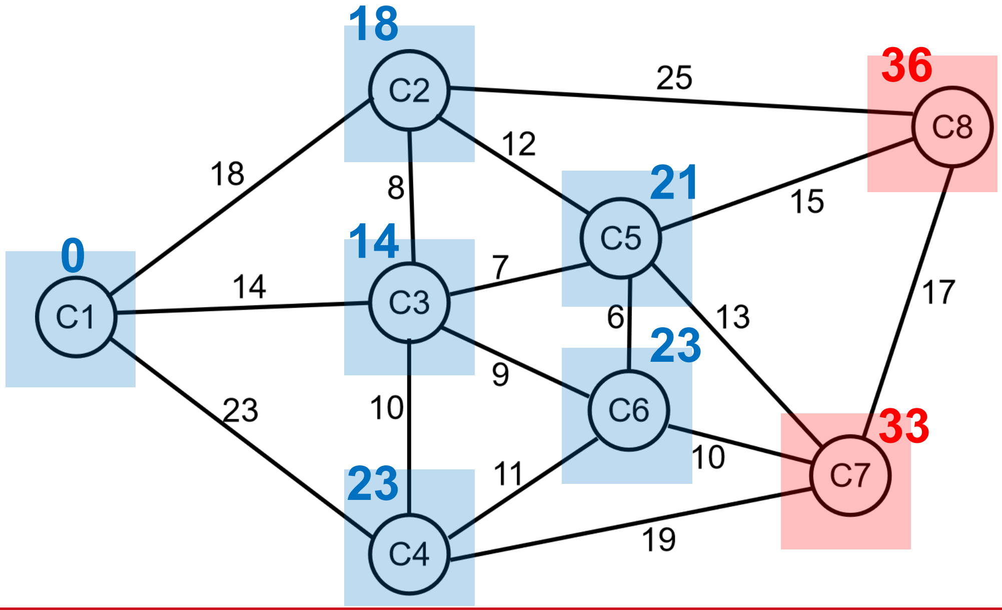
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

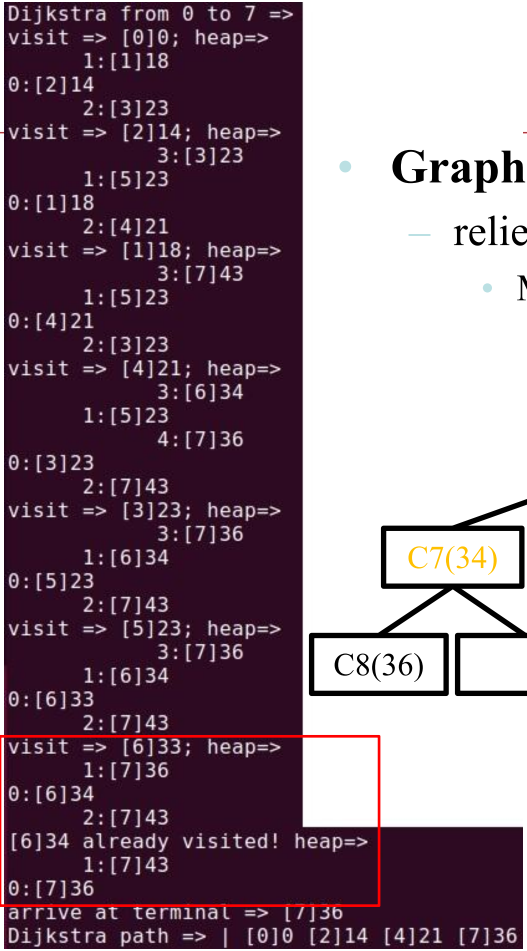
heap : C7 C7 C8 C8



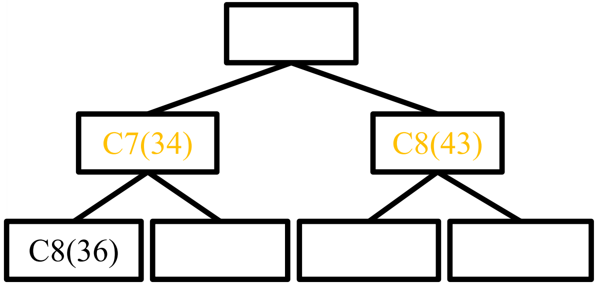


Graph

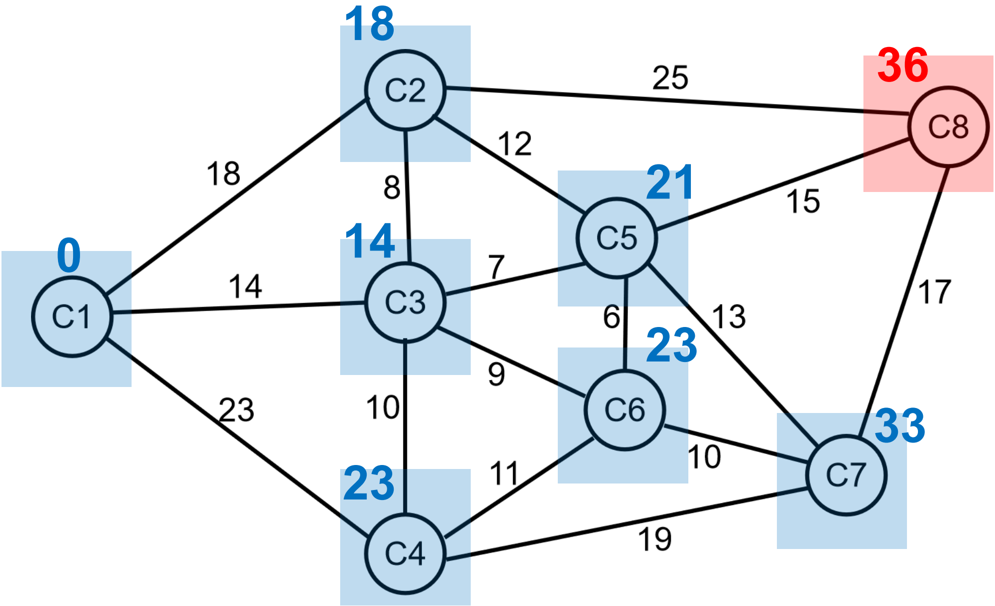
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

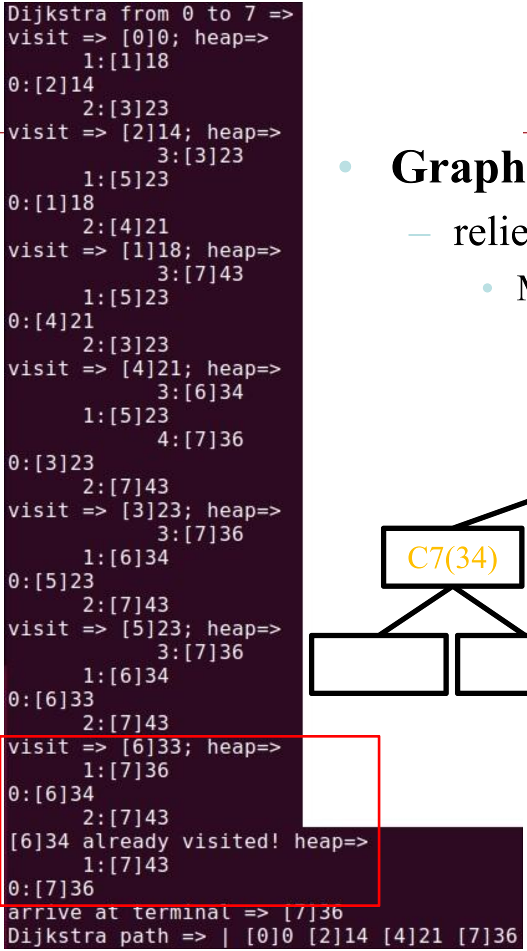


heap : C7 C8 C8

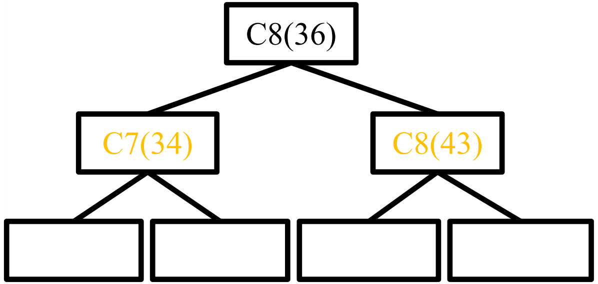


Graph

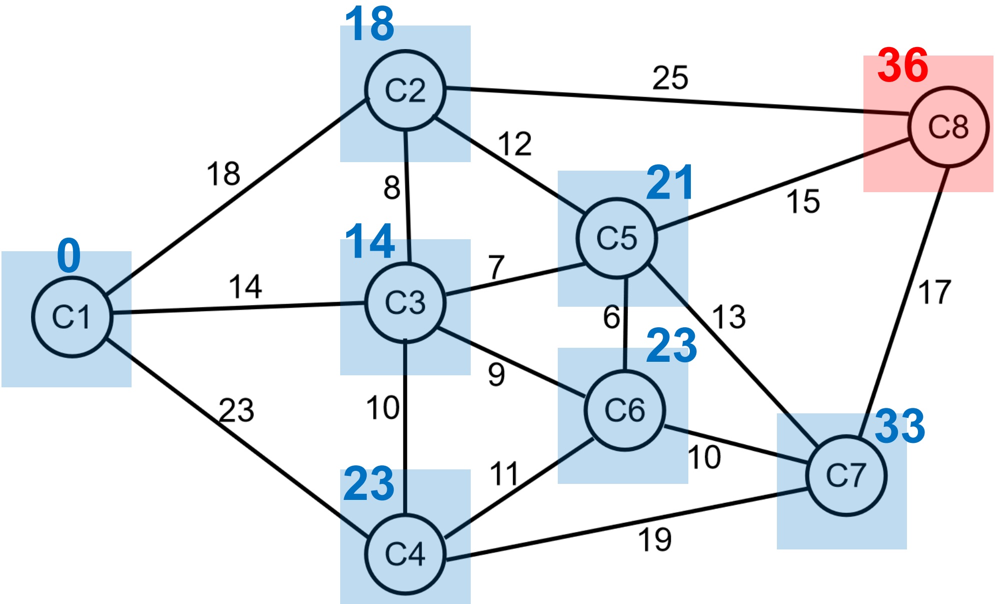
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

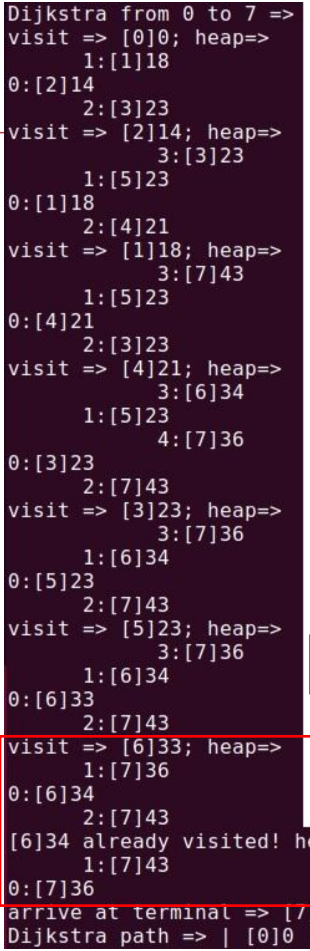


```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```



heap : C8 C7 C8



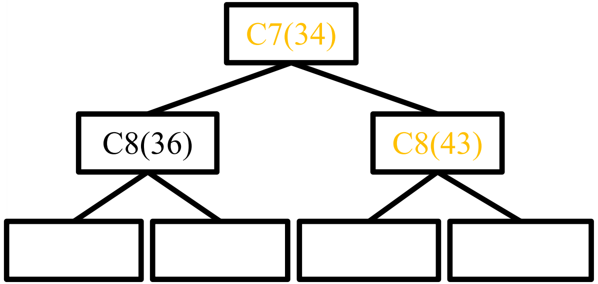


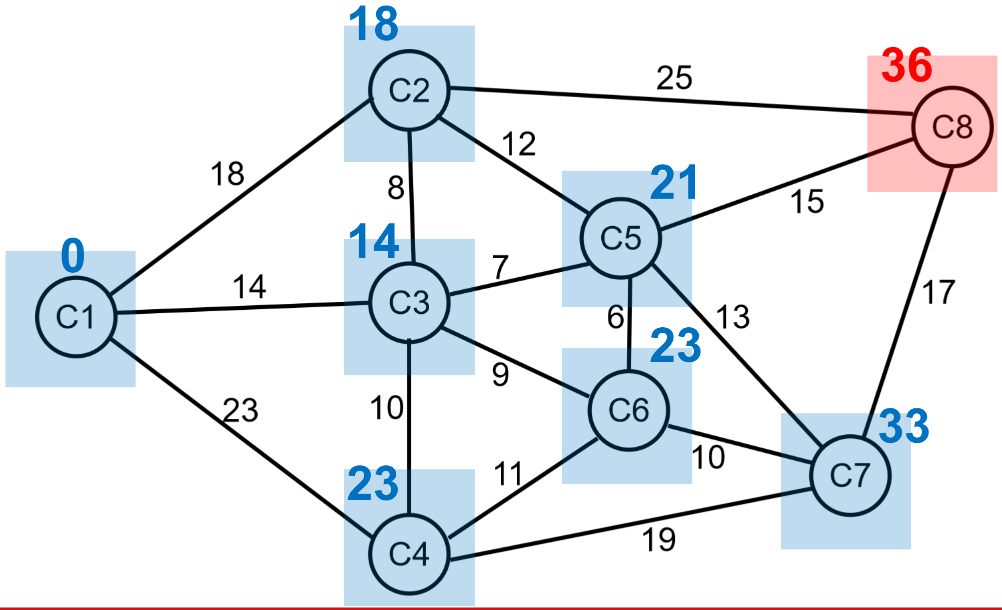
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

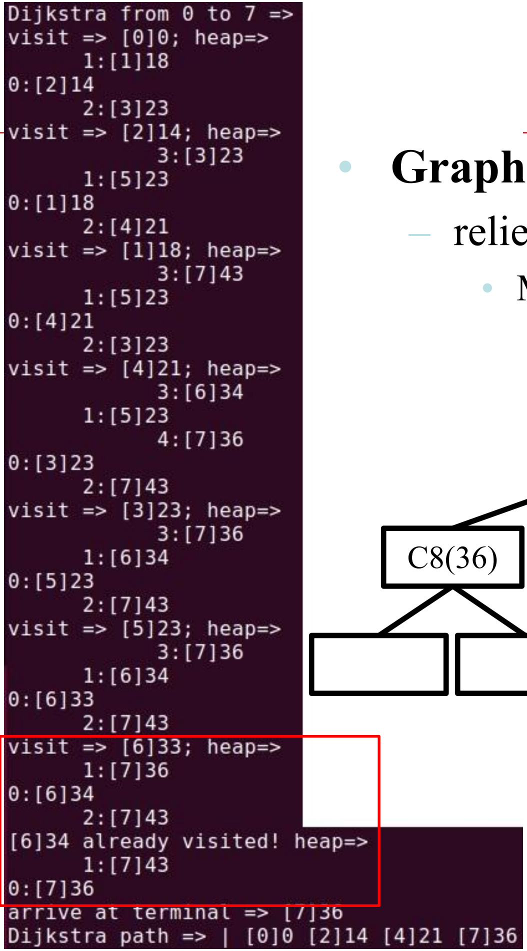
Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap : C7 C8 C8





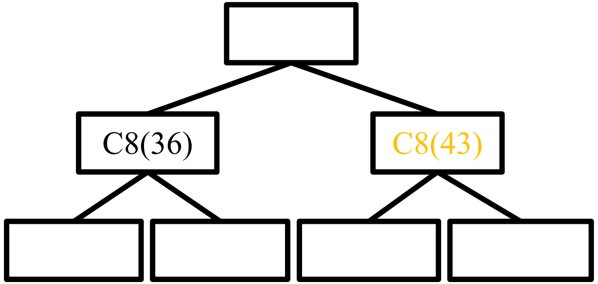


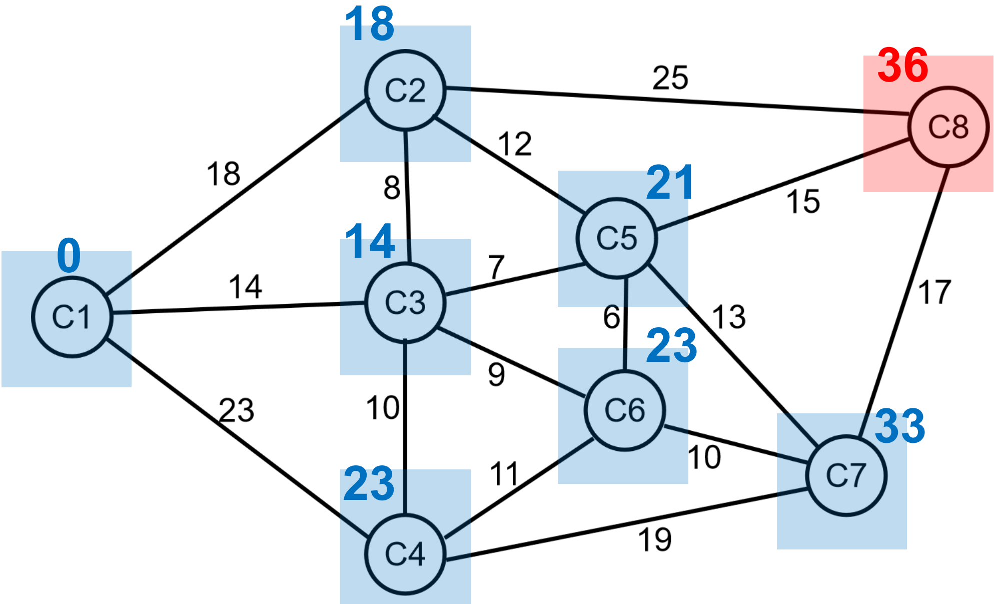
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

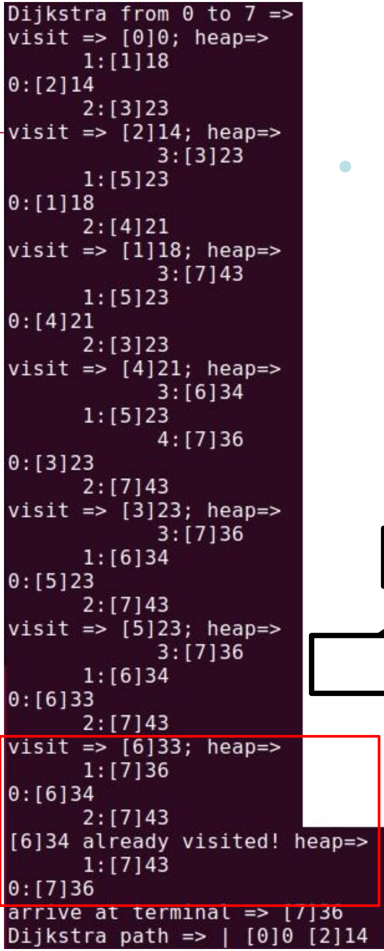
Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap : C8 C8





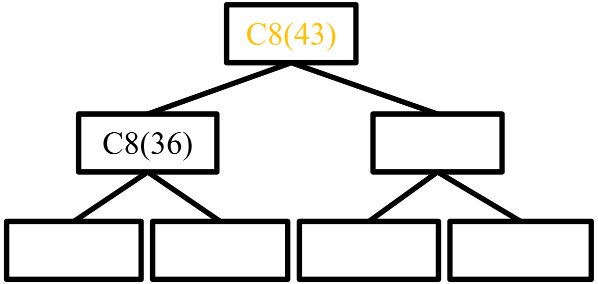


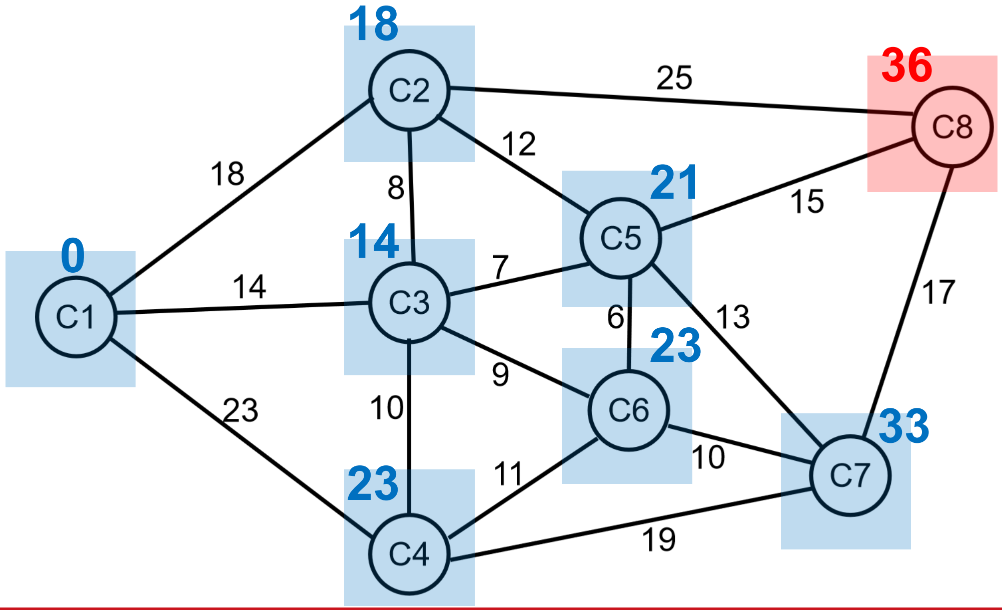
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

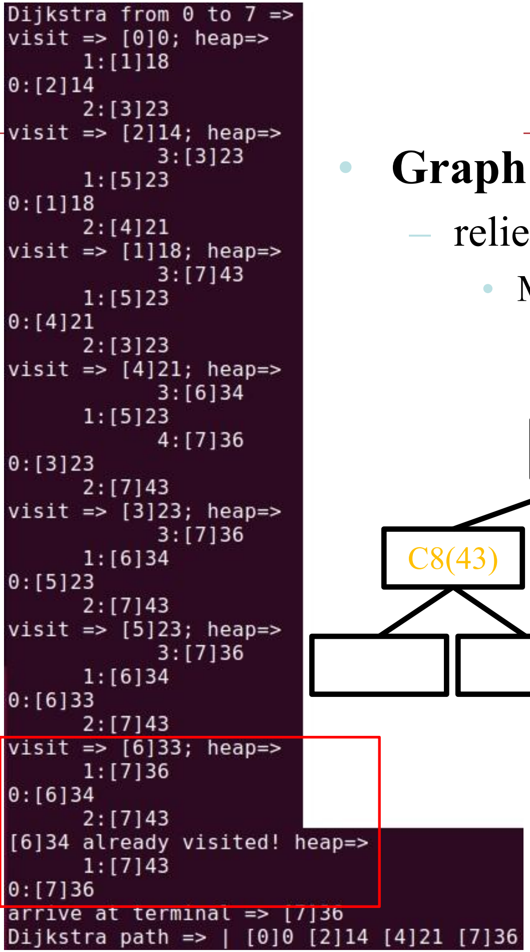
heap : C8 C8



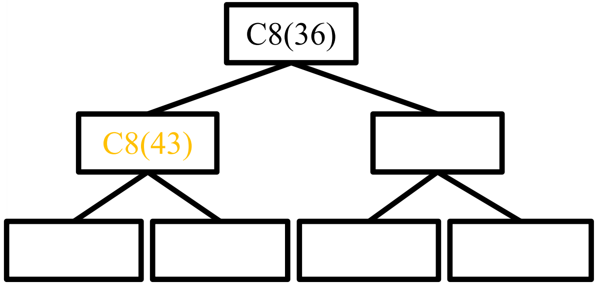


Graph

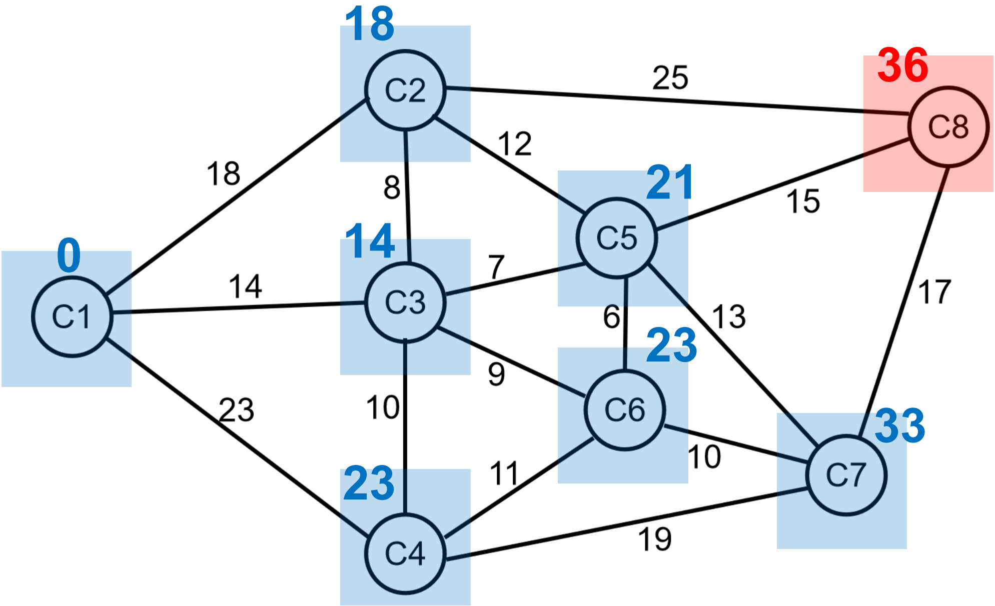
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

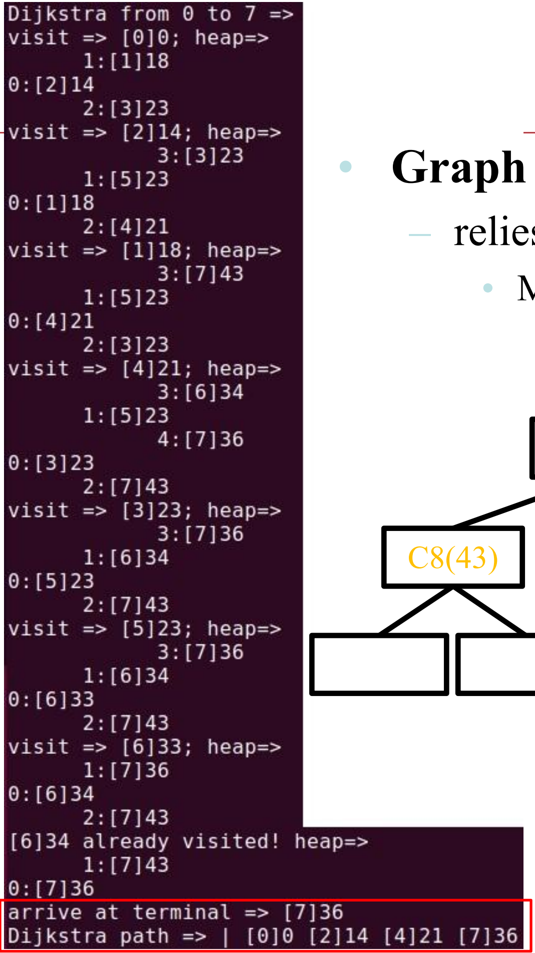


```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```



heap : C8 C8



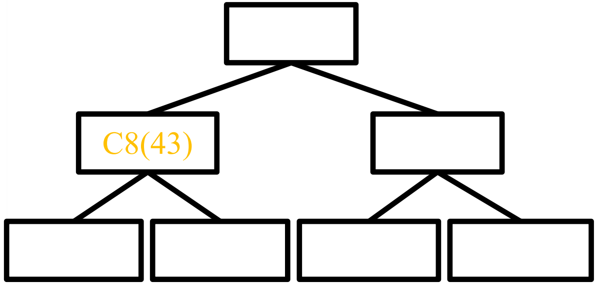


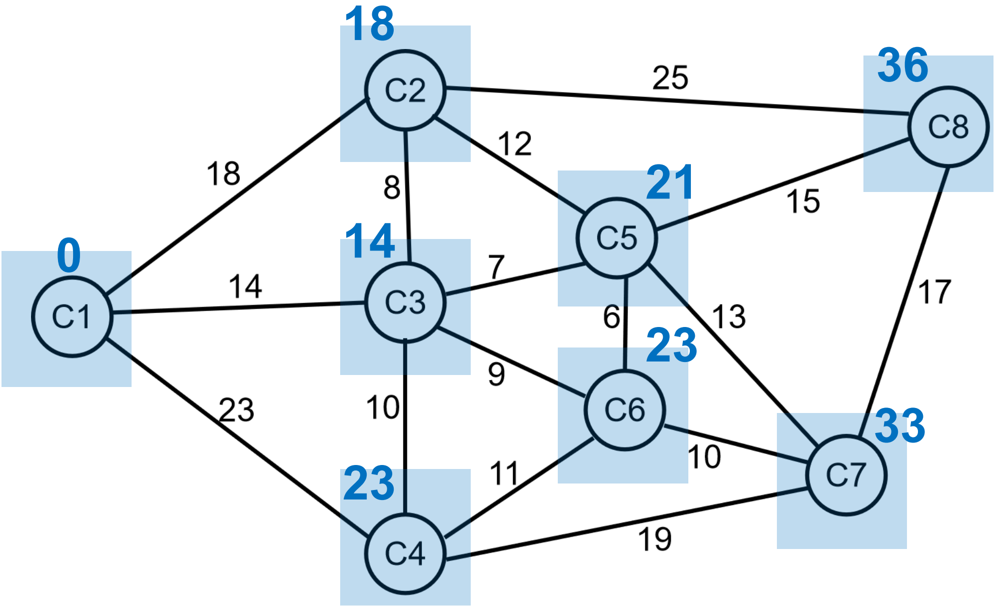
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap : C8

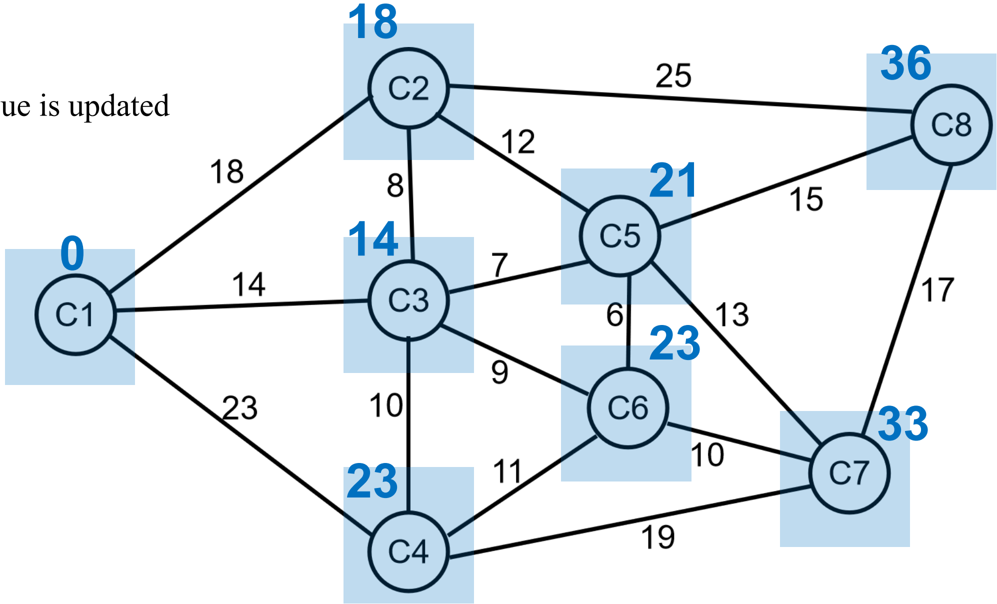




Graph

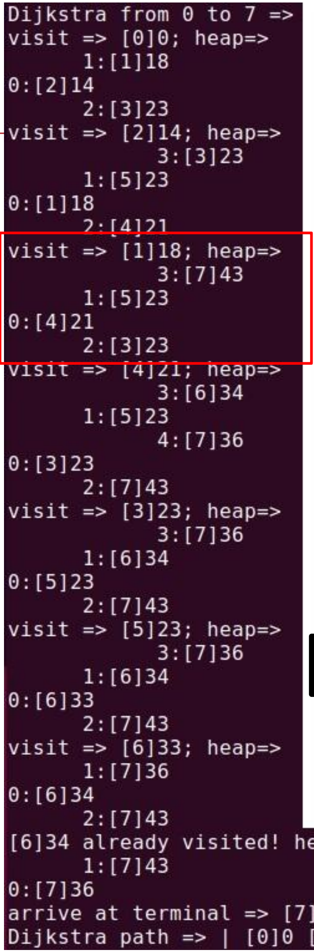
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)
  - retrieve the min-path
    - track the *preceding vertex*
      - from which the vertex value is updated

min-distance(C1,C8) = 36
min-path(C1,C8): **?**

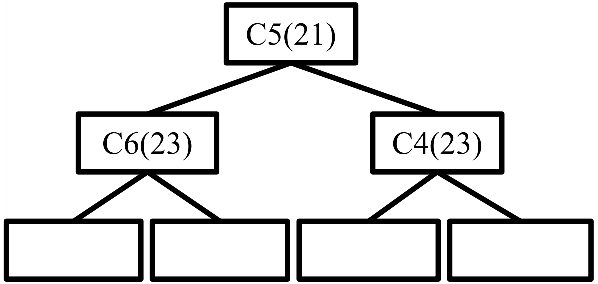


Graph

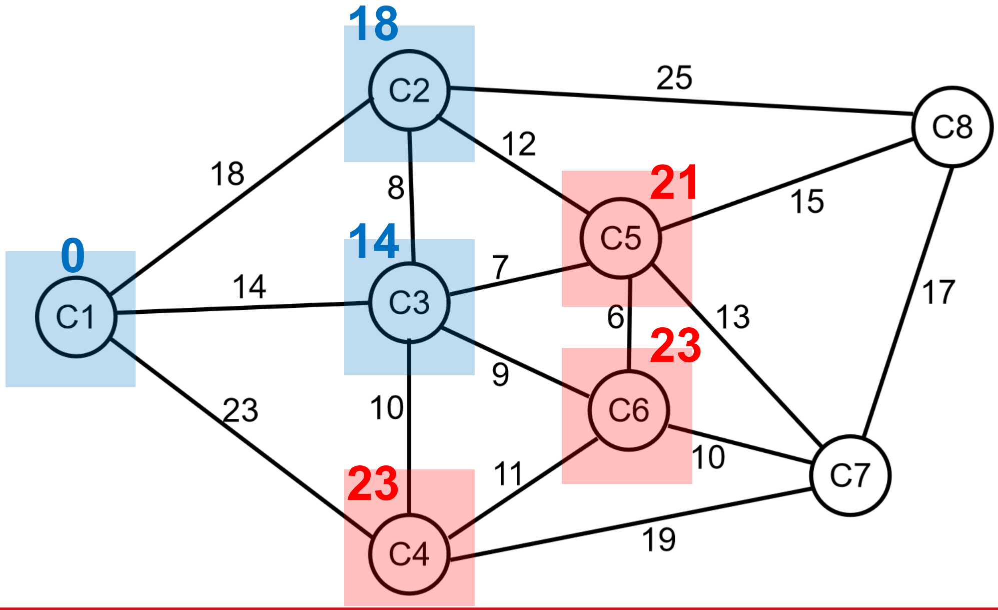
- **Graph - *Dijkstra* algorithm** - single-pair shortest path **REVIEW**
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

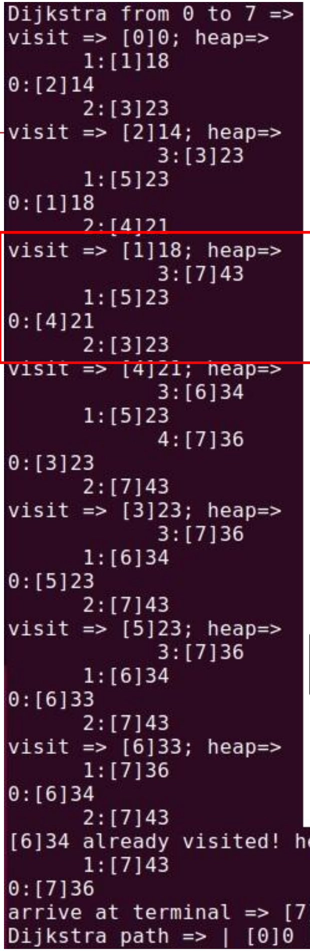


```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```



heap : C5 C6 C4





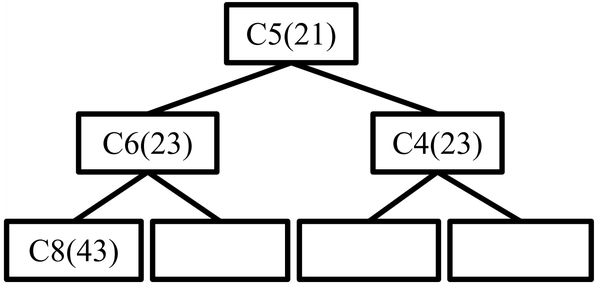
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

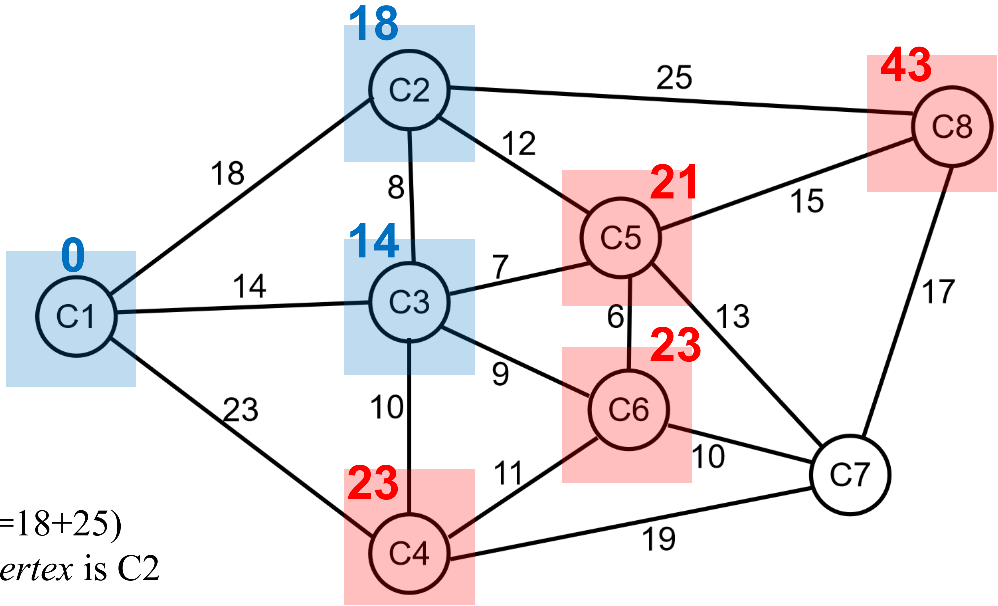
Graph

REVIEW

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap : C5 C6 C4 C8



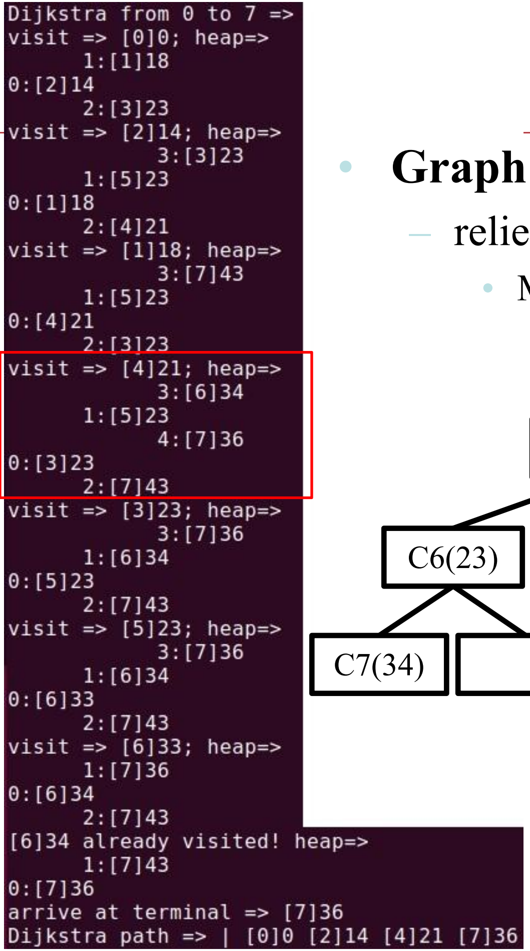


C8 <= C2 (43=18+25)
C8’s *preceding vertex* is C2

Graph

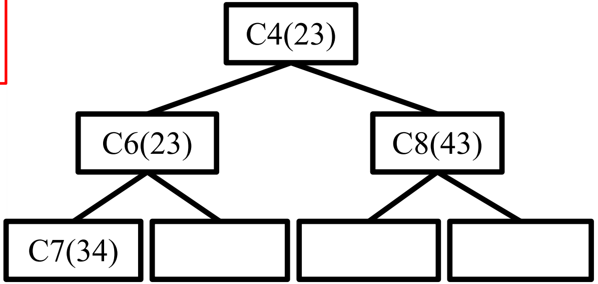
REVIEW

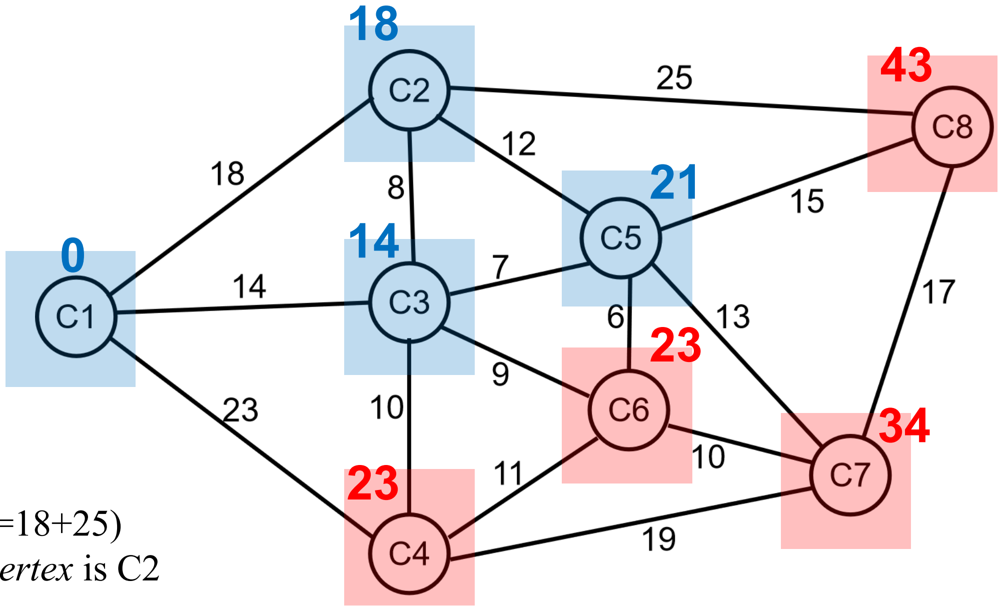
- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)



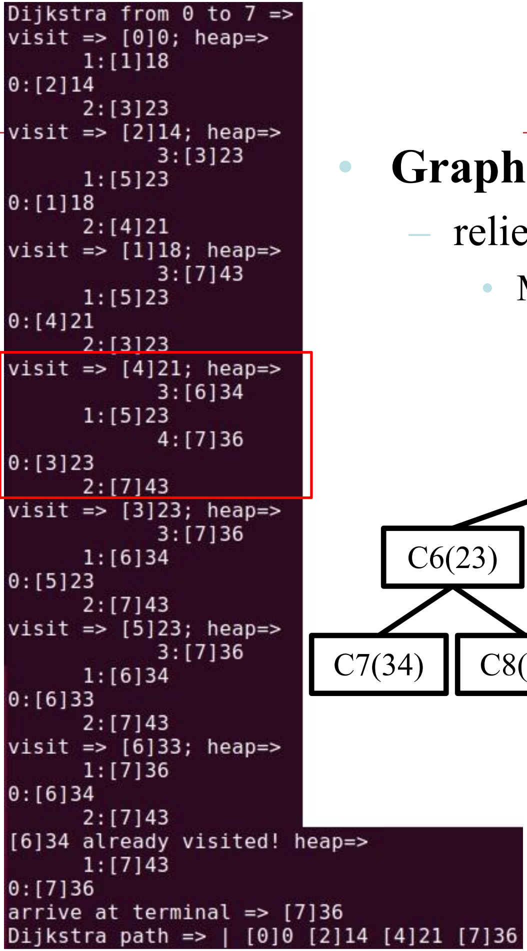
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
        1:[1]18
0:[2]14
        2:[3]23
visit => [2]14; heap=>
                3:[3]23
        1:[5]23
0:[1]18
        2:[4]21
visit => [1]18; heap=>
                3:[7]43
        1:[5]23
0:[4]21
        2:[3]23
visit => [4]21; heap=>
                3:[6]34
        1:[5]23
                4:[7]36
0:[3]23
        2:[7]43
visit => [3]23; heap=>
                3:[7]36
        1:[6]34
0:[5]23
        2:[7]43
visit => [5]23; heap=>
                3:[7]36
        1:[6]34
0:[6]33
        2:[7]43
visit => [6]33; heap=>
        1:[7]36
0:[6]34
        2:[7]43
[6]34 already visited! heap=>
        1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

heap : C4 C6 C8 C7





C8 <= C2 (43=18+25)
C8's *preceding vertex* is C2

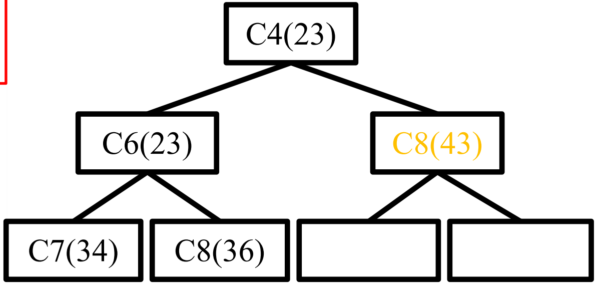


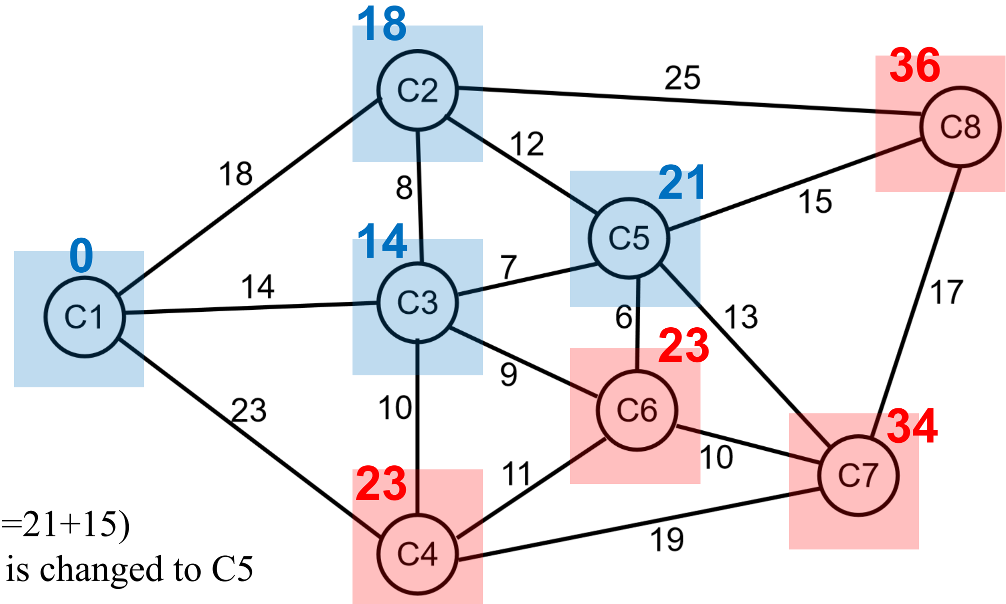
```
Dijkstra from 0 to 7 =>
visit => [0]0; heap=>
      1:[1]18
0:[2]14
      2:[3]23
visit => [2]14; heap=>
            3:[3]23
      1:[5]23
0:[1]18
      2:[4]21
visit => [1]18; heap=>
            3:[7]43
      1:[5]23
0:[4]21
      2:[3]23
visit => [4]21; heap=>
            3:[6]34
      1:[5]23
            4:[7]36
0:[3]23
      2:[7]43
visit => [3]23; heap=>
            3:[7]36
      1:[6]34
0:[5]23
      2:[7]43
visit => [5]23; heap=>
            3:[7]36
      1:[6]34
0:[6]33
      2:[7]43
visit => [6]33; heap=>
      1:[7]36
0:[6]34
      2:[7]43
[6]34 already visited! heap=>
      1:[7]43
0:[7]36
arrive at terminal => [7]36
Dijkstra path => | [0]0 [2]14 [4]21 [7]36
```

Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path **REVIEW**
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)

heap : C4 C6 C8 C7 C8





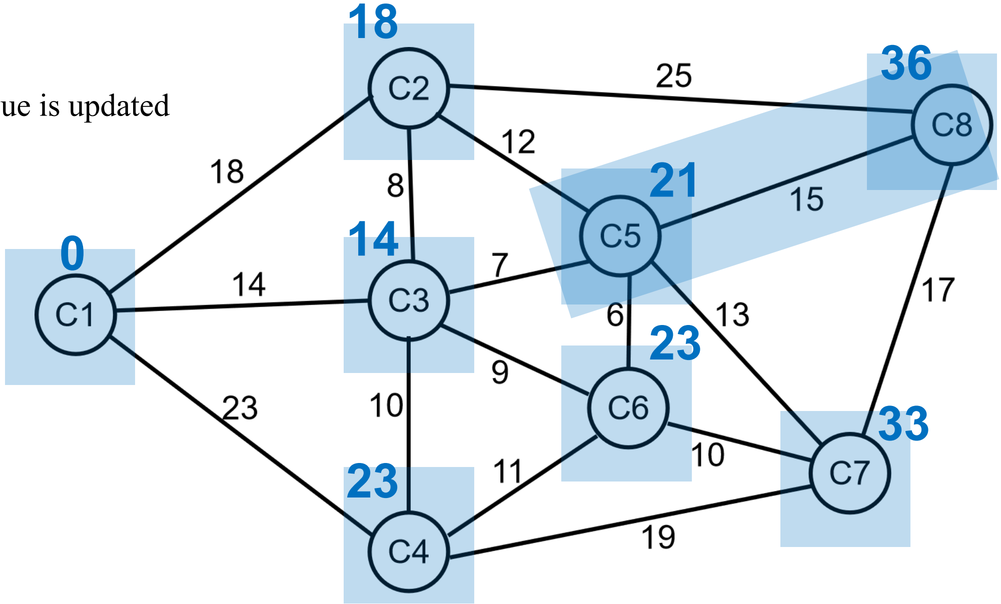
C8 <= C5 (36=21+15)

C8's *preceding vertex* is changed to C5

Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)
  - retrieve the min-path
    - track the *preceding vertex*
      - from which the vertex value is updated

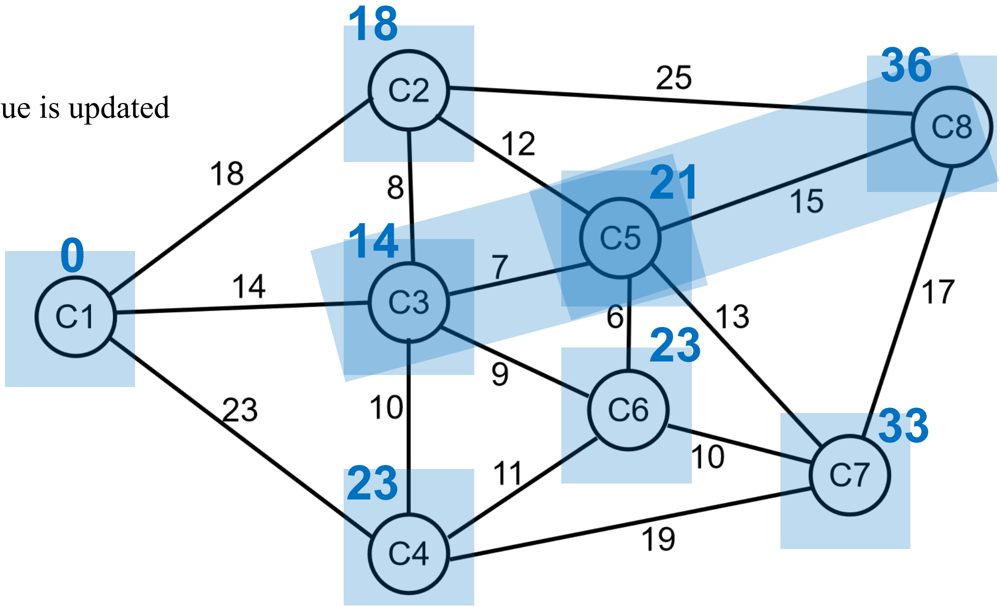
min-distance(C1,C8) = 36
min-path(C1,C8): C5=>C8
C8's preceding vertex is C5



Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)
  - retrieve the min-path
    - track the *preceding vertex*
      - from which the vertex value is updated

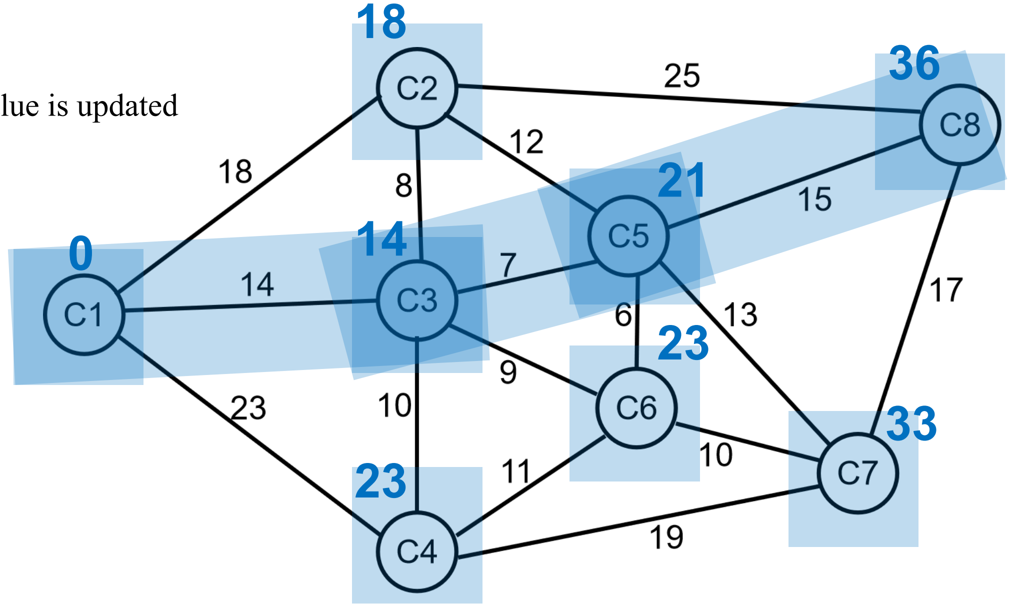
min-distance(C1,C8) = 36
min-path(C1,C8): C3=>C5=>C8
C5's preceding vertex is C3



Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)
  - retrieve the min-path
    - track the *preceding vertex*
      - from which the vertex value is updated

min-distance(C1,C8) = 36
min-path(C1,C8): C1=>C3=>C5=>C8
C3's preceding vertex is C1



Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)
  - retrieve the min-path - track the *preceding vertex*
  - complexity
    - adjacency list based *Dijkstra* with heap based MDFO: O((|V|+|E|) *log* |V|)
      - each vertex involves one *heap removeroot*: O(*log* |V|)
      - each vertex involves E[*out degree*] times of potential *heap insert*: O(E[*out degree*] *log* |V|)
      - totally |V| O(*log* |V| + E[*out degree*] *log* |V|) = O(|V| *log* |V|) + O(|E| *log* |V|)
        - » |E| is the number of directed edges; an undirected edge counts as two directed edges
    - adjacency matrix based *Dijkstra*: O($|V|^2$)
      - *direct vertex traversal* based MDFO
      - each vertex involves |V| times of distance checking: O(|V|)
      - each vertex involves |V| times of potential neighbourhood update: O(|V|)
      - totally |V| O(|V| + |V|) = O($|V|^2$)

Graph

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure (normally a *heap*) that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)
  - retrieve the min-path - track the *preceding vertex*
  - complexity
    - adjacency list based *Dijkstra* with heap based MDFO: O((|V|+|E|) *log* |V|)
    - adjacency matrix based *Dijkstra*: O($|V|^2$)
    - critical point: E[*out degree*] = |V| / log |V|, when O((|V|+|E|) *log* |V|) = O($|V|^2$)
      - e.g. given a graph whose vertices are indexed 0,1,...,n-1 (suppose n is large enough), for a generic vertex k, a directed edge connects vertex k to vertex (k+d)%n, if & only if d is a prime number < n; such a graph has each vertex's out degree expectation ≈ |V| / log |V|.
      - reflect: in your opinion, such a graph is sparse or dense?

adjacency list representation
0=>[2,3,5,7,11,13,...]
1=>[3,4,6,8,12,14,...]
2=>[4,5,7,9,13,15,...]
... ...
n-1=>[1,2,4,6,10,12,...]

Graph

REVIEW

- **Graph - *Dijkstra* algorithm** - single-pair shortest path
  - relies on a structure that performs MDFO instead of FIFO
    - MDFO (minimum distance first out)
  - *sparse graphs* normally resort to a *heap* for efficient MDFO implementation
    - sparse graphs are much more common than dense graphs in practical applications
    - sparse graphs normally adopt adjacency list representation
    - heap based MDFO is dedicated to sparse graphs that adopt adjacency list representation
  - *dense graphs* normally resort to *direct vertex traversal* for MDFO implementation
    - heap based MDFO brings no benefit to dense graphs
    - dense graphs are much less common than sparse graphs in practical applications

believe now you can understand why

THANK YOU

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY