05 Graph (4)

College of Computer Science, CQU

Outline

- Minimum-Cost Spanning Trees
- Prim's Algorithm
- Kruskal's Algorithm

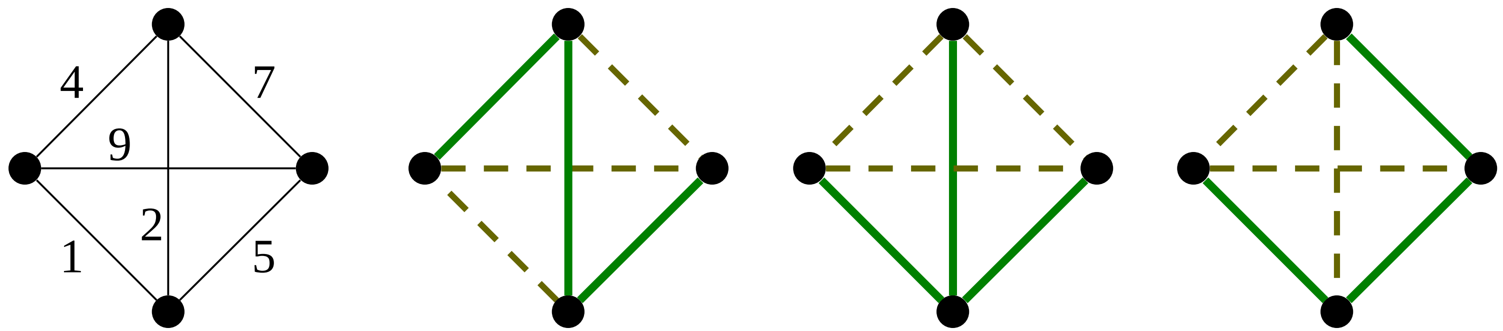
Knowledge Points

- Chapter 11, pp.402-409

Spanning Tree

- **Spanning tree** - a subset of the edges from a connected graph that:
  - touches all vertices in the graph (**spans** the graph)
  - forms a tree (is connected and contains no cycles)

- **Minimum spanning tree** - spanning tree with lowest total edge cost



Prim’s Algorithm(扩点法)

- Prim’s Algorithm (a variation of Dijkstra’s Algorithm) can finds Minimum Spanning Trees:
  - Pick an initial node
  - Until graph is connected:

    Choose edge (u,v) which is of minimum cost among edges where u is in tree but v is not

    Add (u,v) to the tree

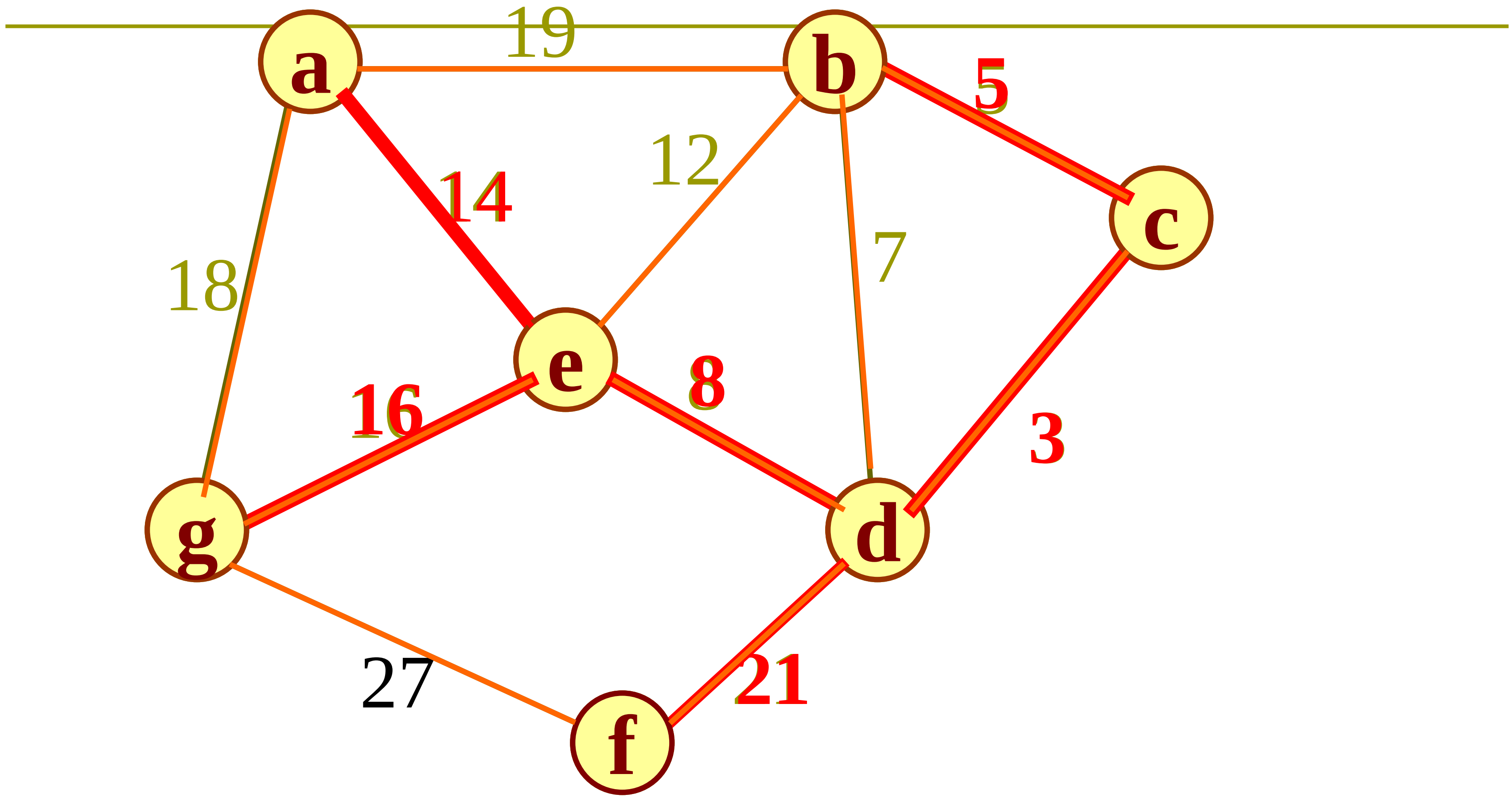
- Same “greedy” proof, same asymptotic complexity

Prim’s Algorithm(扩点法)

Process:

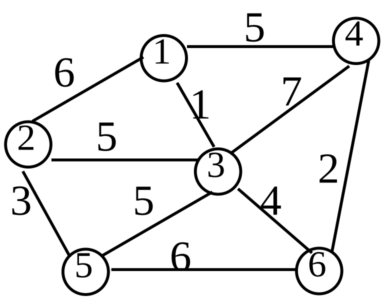
```
T={ };
TV = { 0 }; // start with vertex 0 and no edges
 while (T contains fewer than n-1 edges) {
       let (u,v) be a least cost edge such that
             u∈TV and not v∈TV;
       if (there is no such edges) break;
       add v into Tv;
       add (u, v) into T;
 }
if (T contains fewer than n-1 edges)
       print("No spanning tree");
```

例如:

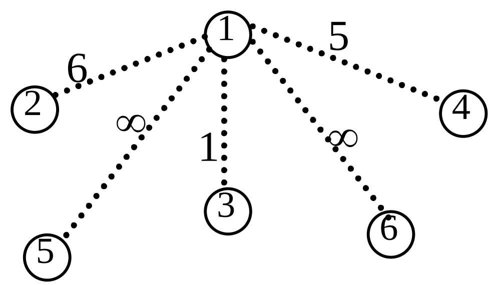


所得生成树权值和 $= 14+8+3+5+16+21 = 67$

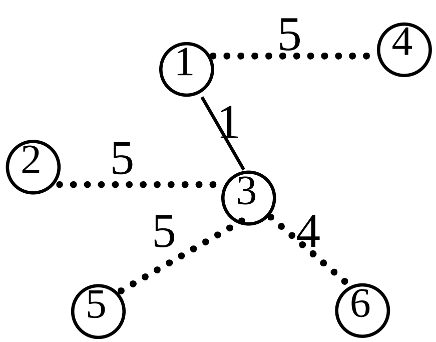
假设开始顶点就选为顶点1，故首先有U={1}，W=V-U={2，3，4，5，6}



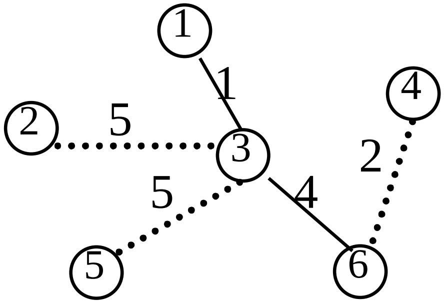
(a)无向网



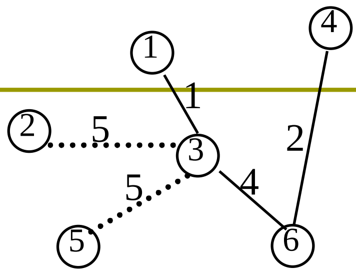
(b) u={1} w={2,3,4,5,6}

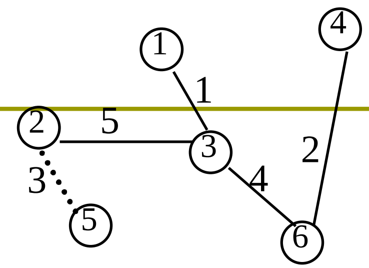


(c) u={1,3} w={2,4,5,6}

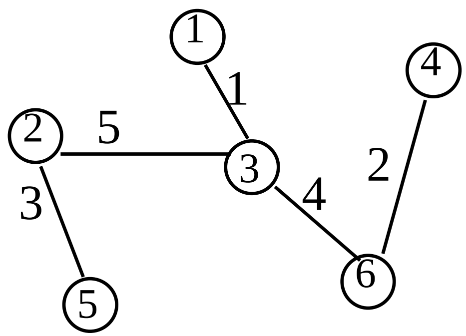


(d) u={1,3,6} w={2,4,5}





(e) u={1,3,6,4}　w={2,5}　(f) u={1,3,6,4,2}　　w={5}



(g) u={1,3,6,4,2,5}　w={ }

prim 方法构造最小生成树的过程

Prim’s Algorithm(扩点法)

```
void Prim(Graph* G, int* D, int s) { // Prim's MST algorithm
  int V[G->n()];                       // Store closest vertex
  int i, w;
  for (i=0; i<G->n(); i++) {           // Process the vertices
    int v = minVertex(G, D);
    G->setMark(v, VISITED);
    if (v != s)
      AddEdgetoMST(V[v], v);           // Add edge to MST
    if (D[v] == INFINITY) return;      // Unreachable vertices
    for (w=G->first(v); w<G->n(); w = G->next(v,w))
      if (D[w] > G->weight(v,w)) {
        D[w] = G->weight(v,w);         // Update distance
        V[w] = v;                      // Where it came from
      }
  }
}
```

Prim's Algorithm(扩点法)

```
// Prim's MST algorithm: priority queue version
void Prim(Graph* G, int* D, int s) {
  int i, v, w;              // "v" is current vertex
  int V[G->n()];            // V[I] stores I's closest neighbor
  DijkElem temp;
  DijkElem E[G->e()];       // Heap array with lots of space
  temp.distance = 0; temp.vertex = s;
  E[0] = temp;              // Initialize heap array
  heap<DijkElem, DDComp> H(E, 1, G->e()); // Create heap
```

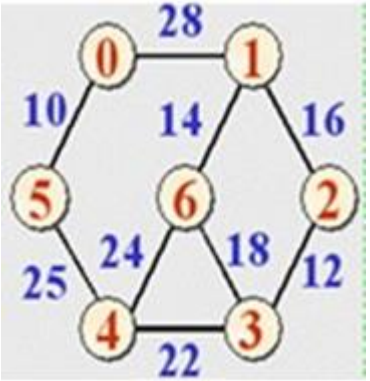
Prim’s Algorithm(扩点法)

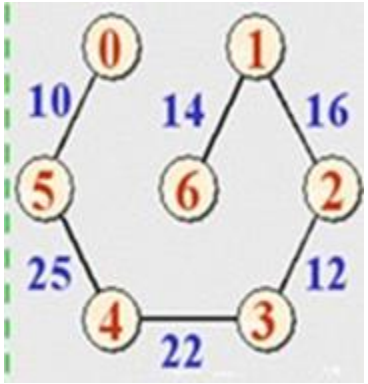
```
for (i=0; i<G->n(); i++) {                // Now build MST
  do {
    if(H.size() == 0) return; // Nothing to remove
    temp = H.removefirst();
    v = temp.vertex;
  } while (G->getMark(v) == VISITED);
  G->setMark(v, VISITED);
  if (v != s) AddEdgetoMST(V[v], v); // Add edge to MST
  if (D[v] == INFINITY) return;       // Ureachable vertex
```

Prim's Algorithm(扩点法)

```
  for (w=G->first(v); w<G->n(); w = G->next(v,w))
    if (D[w] > G->weight(v, w)) {     // Update D
      D[w] = G->weight(v, w);
      V[w] = v;               // Update who it came from
      temp.distance = D[w]; temp.vertex = w;
      H.insert(temp);   // Insert new distance in heap
    }
  }
}
```

Exercise of Prim's Algorithm





Homework

- Reading programs:
- 1.grmat.h grlist.h heap.h
- 2.grpriml1.cpp grpriml2.cpp
- 3. grprimm1.cpp grprimm2.cpp

Kruskal's Algorithm (扩边法)

- Yet another greedy algorithm

- Initialize all vertices to unconnected
- While there are still unmarked edges
  - Pick the lowest cost edge **`e = (u, v)`** and mark it
  - If **`u`** and **`v`** are not already connected, add **`e`** to the minimum spanning tree and connect **`u`** and **`v`**

- How is this like maze generation?
- How is it different?

Kruskal’s Algorithm (扩边法)

- Yet another greedy algorithm

- Partition the set of vertices into |V| equivalence classes;
- Process the edges in order of weight;
- While (edgecount<n-1)
  - Pick the lowest cost edge **`e = (u, v)`**
  - If after the edge e is added to the MST, there is no cycle(if the edge connected two vertertices in different equivalence), then add **`e`** to the minimum spanning tree and connect **`u`** and **`v, or discard e`**

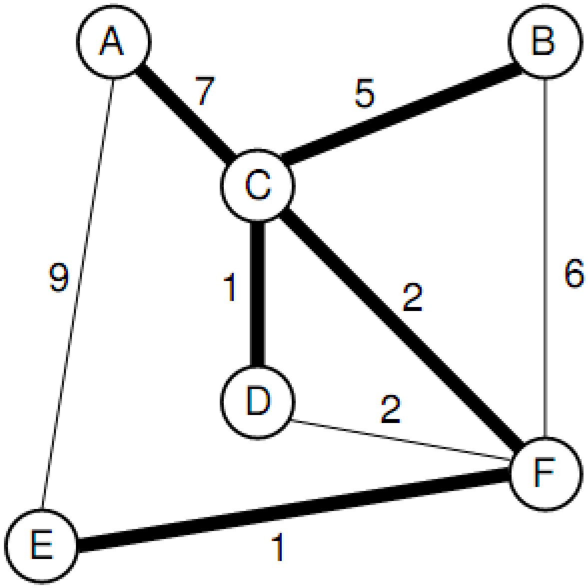
- How is this like maze generation?
- How is it different?

Kruskal's Algorithm

Algorithm:

```
T={ };
while (T contains less than n-1 edges &&
            E is not empty ){
        Choose a least cost edge (v,w) from E;
        delete (v,w) from E;
        if ((v,w) does not create a cycle in T)
            add (v,w) into T;
        else discard (v,w);
}
if (T contains fewer than n-1 edges)
        print ("No spanning tree\n");
```

Kruskal’s Algorithm



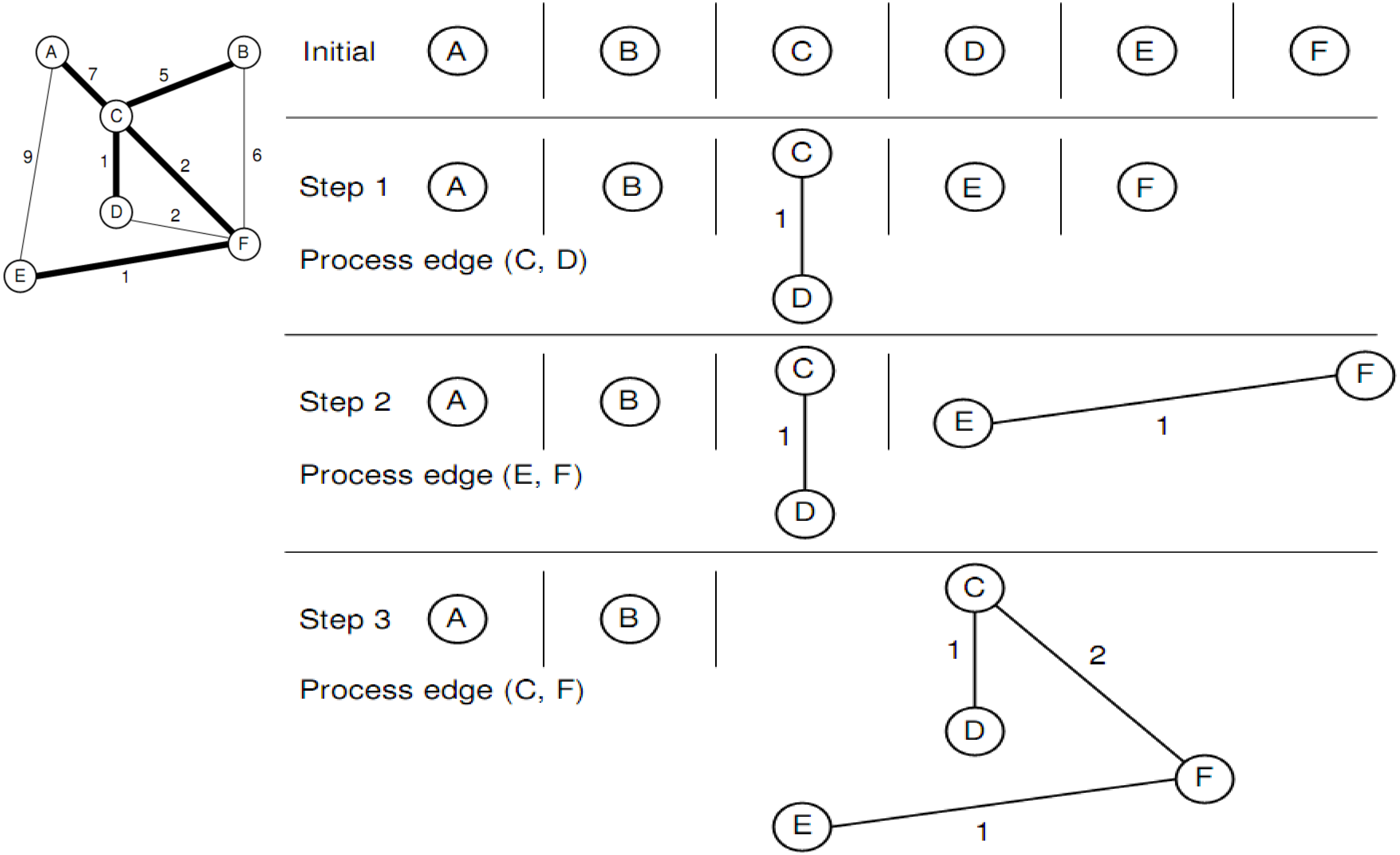
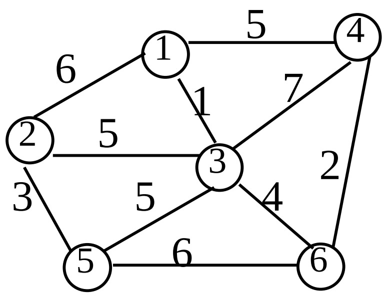
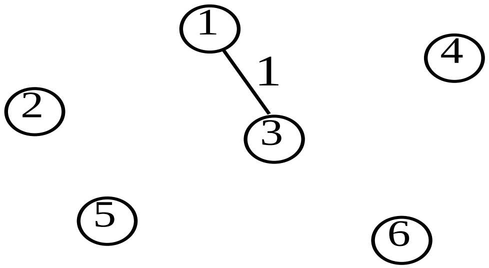


Figure 11.24 Illustration of the first three steps of Kruskal's MST algorithm as applied to the graph of Figure 11.20.

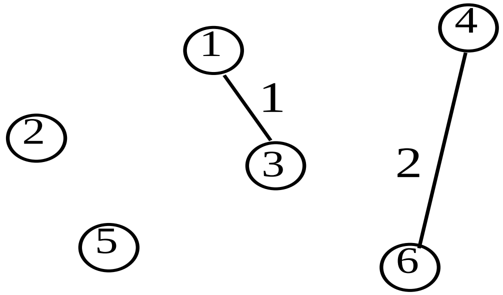
先将图中所有边按权值递增顺序排列，依次选定取权值较小的边，但要求后面选取的边不能与前面选取的边构成回路，若构成回路，则放弃该条边，再去选后面权值较大的边，**n**个顶点的图中，选够**n-1**条边即可。



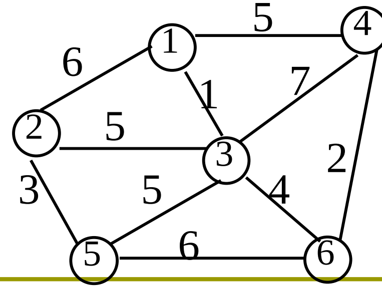
(a)无向网



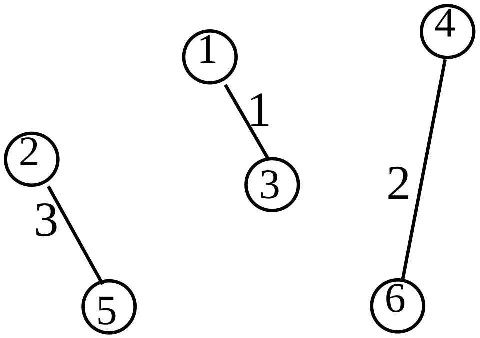
(a) 选第 1 条边



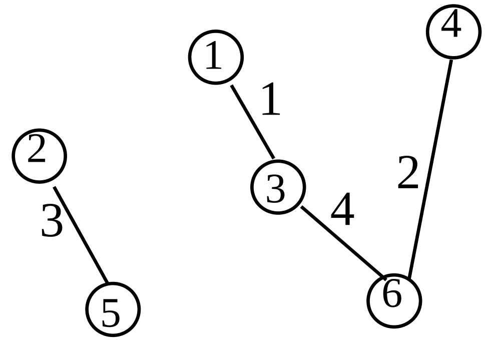
(b) 选第 2 条边



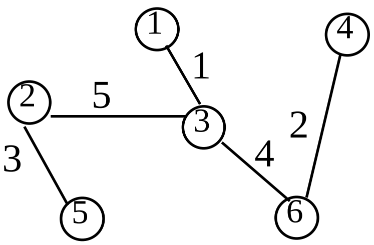
(a)无向网



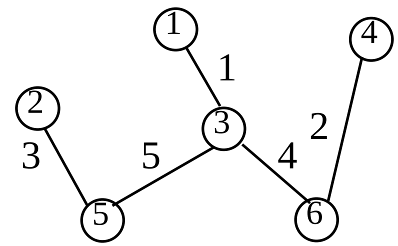
(c) 选第 3 条边



(d) 选第 4 条边



或者



(e) 选第 5 条边(不能选（1,4）边，会构成回路，但可选（2,3）或(5,3)中之一)

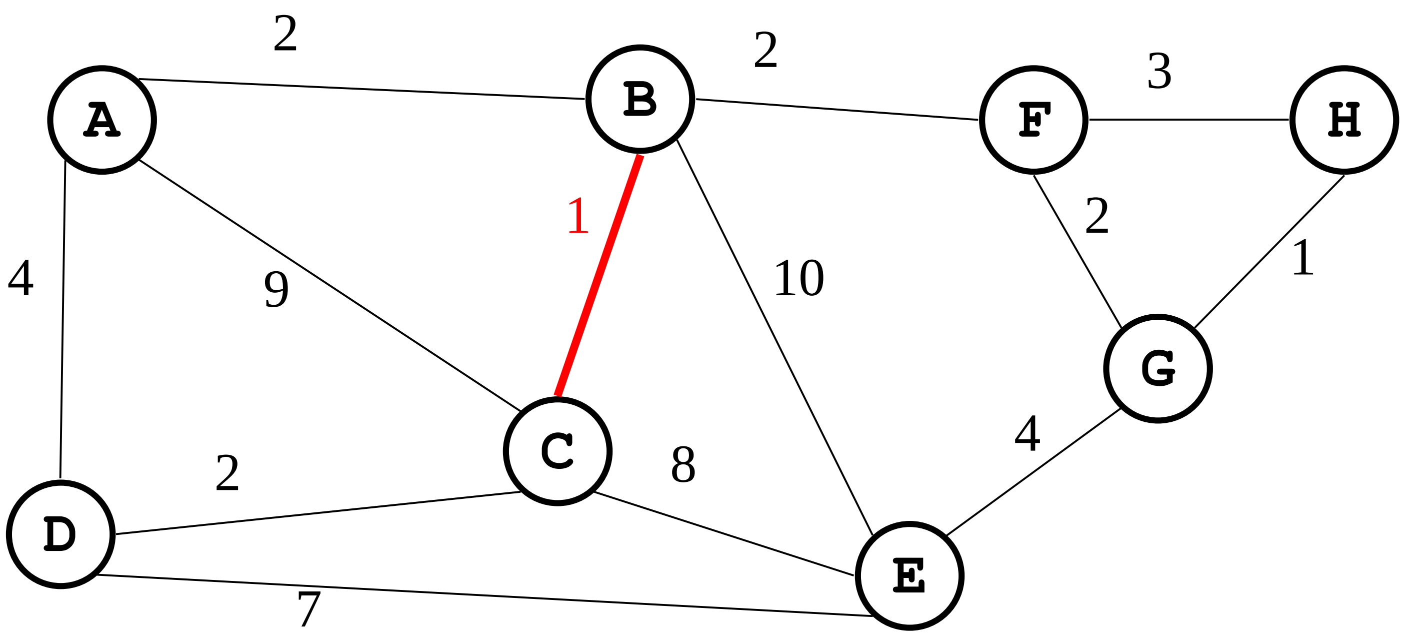
克鲁斯卡尔方法求最小生成树的过程

```
class KruskElem {            // An element for the heap
public:
  int from, to, distance; // The edge being stored
  KruskElem() { from = -1;  to = -1; distance = -1; }
  KruskElem(int f, int t, int d)
    { from = f; to = t; distance = d; }
};
```

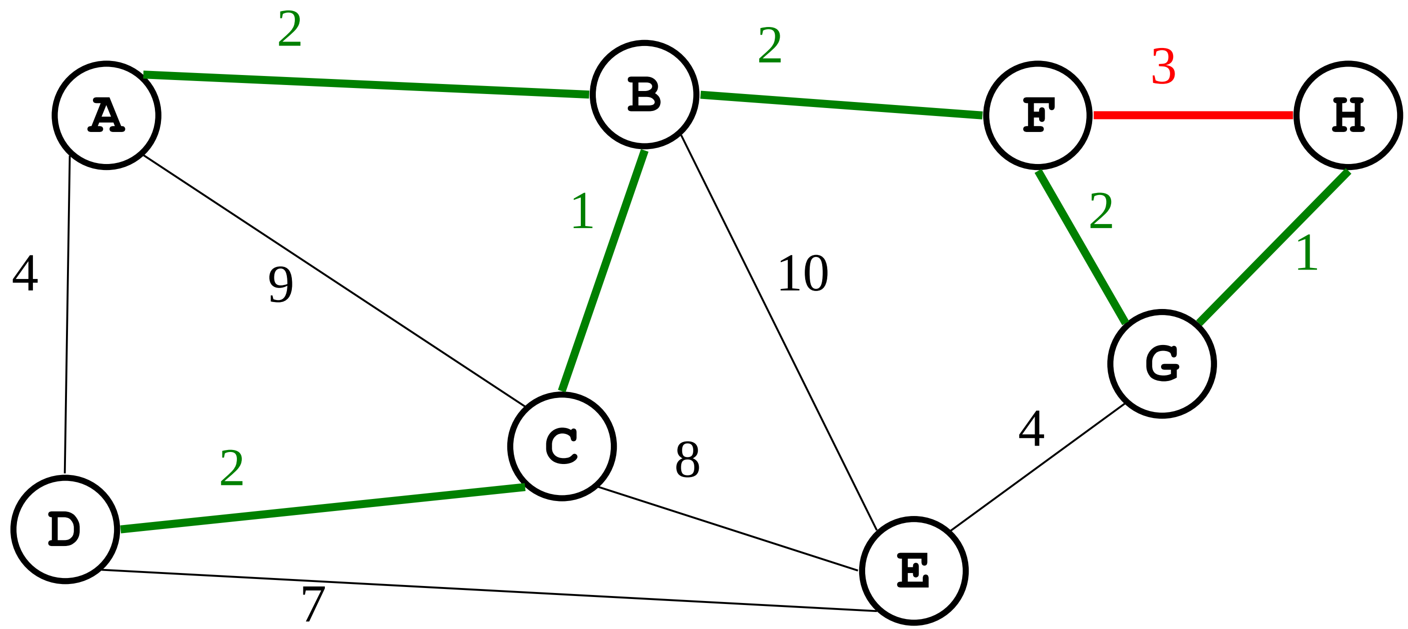
```
void Kruskel(Graph* G) {   // Kruskal's MST algorithm
  ParPtrTree A(G->n());    // Equivalence class array
  KruskElem E[G->e()];     // Array of edges for min-heap
  int i;
  int edgecnt = 0;
  for (i=0; i<G->n(); i++) // Put the edges on the array
    for (int w=G->first(i); w<G->n(); w = G->next(i,w)) {
      E[edgecnt].distance = G->weight(i, w);
      E[edgecnt].from = i;
      E[edgecnt++].to = w;
    }
```

```
  // Heapify the edges
  heap<KruskElem, Comp> H(E, edgecnt, edgecnt);
  int numMST = G->n();          // Initially n equiv classes
  for (i=0; numMST>1; i++) { // Combine equiv classes
    KruskElem temp;
    temp = H.removefirst(); // Get next cheapest edge
    int v = temp.from;  int u = temp.to;
    if (A.differ(v, u)) {  // If in different equiv classes
      A.UNION(v, u);          // Combine equiv classes
      AddEdgetoMST(temp.from, temp.to);  // Add edge to MST
      numMST--;               // One less MST
    }
  }
}
```

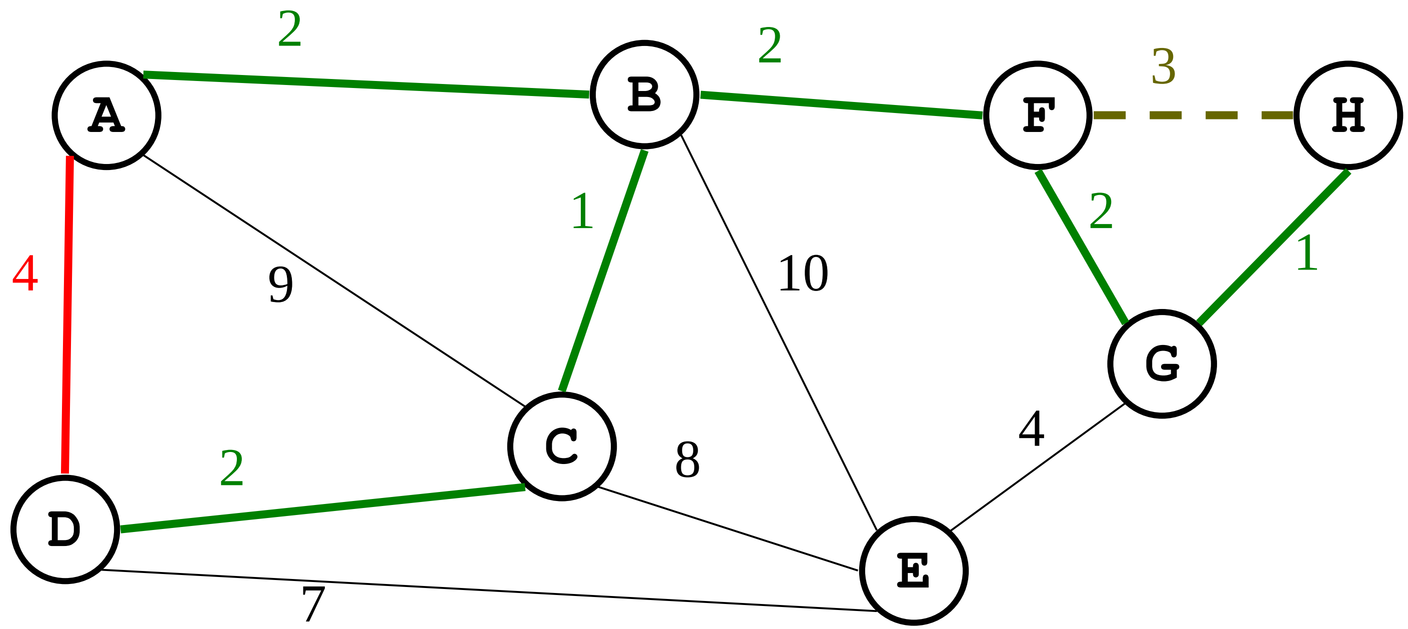
Kruskal's Algorithm in Action (1/5)



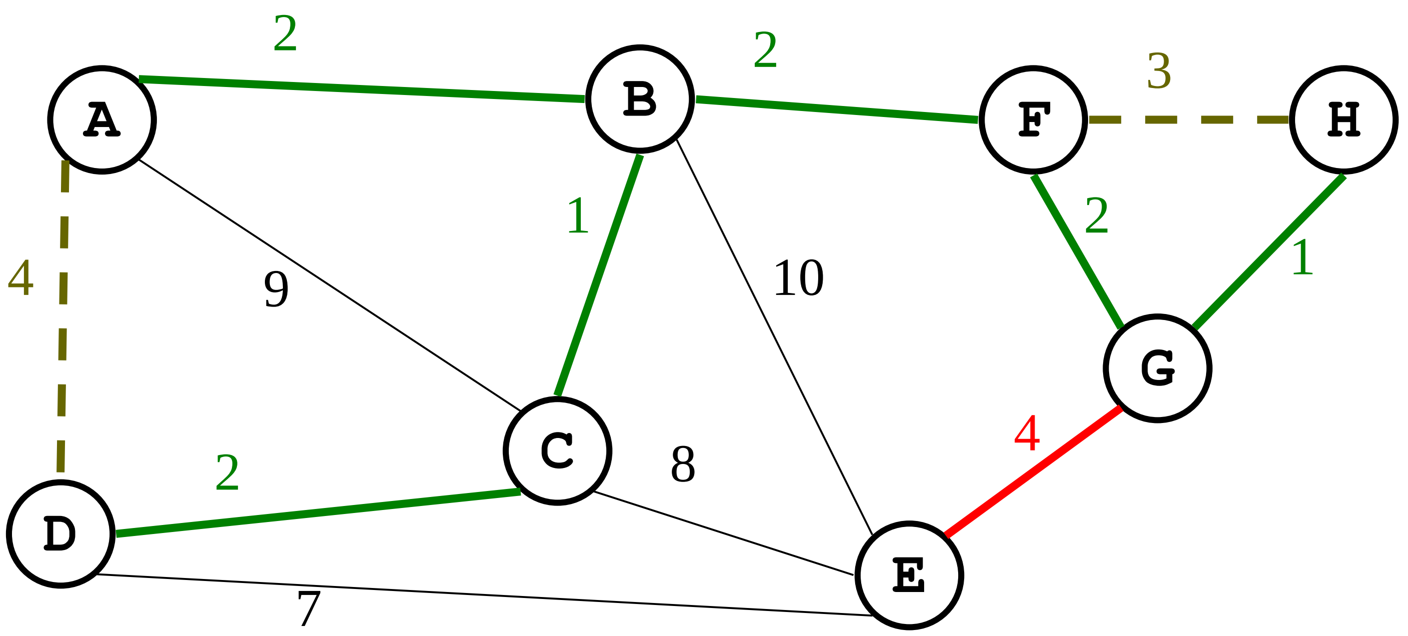
Kruskal's Algorithm in Action (2/5)



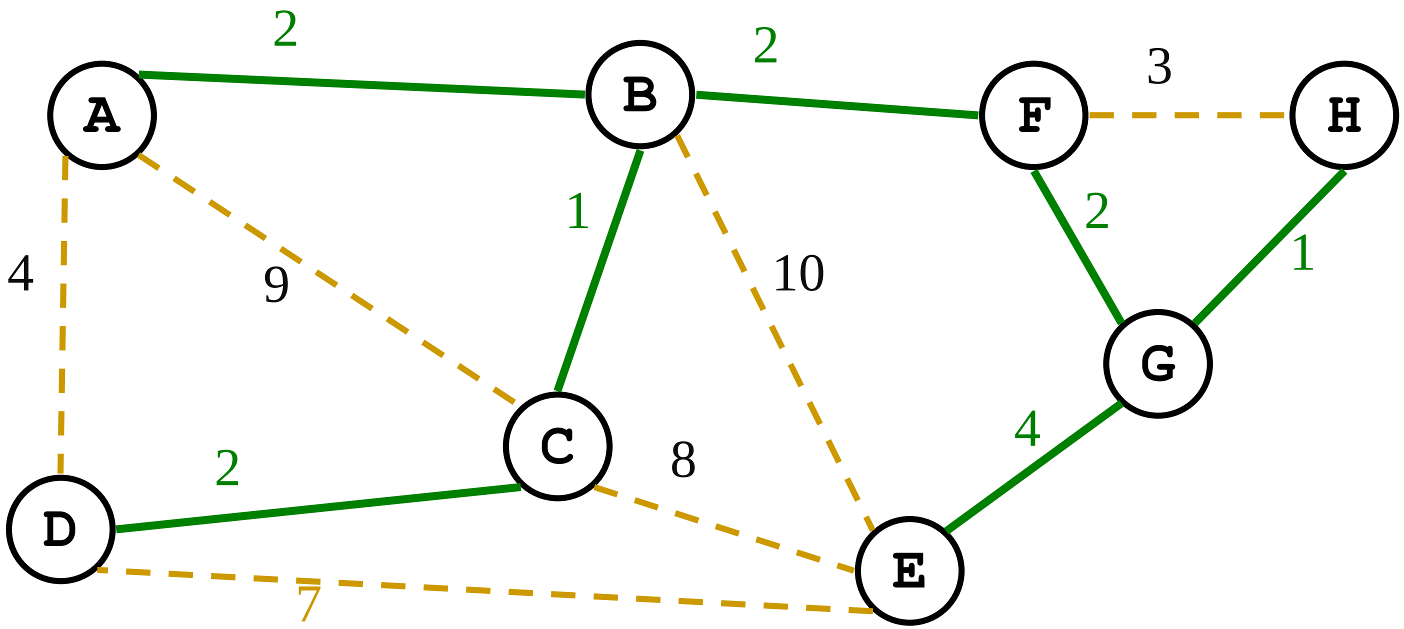
Kruskal's Algorithm in Action (3/5)



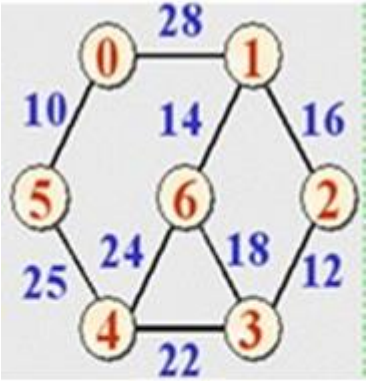
Kruskal's Algorithm in Action (4/5)



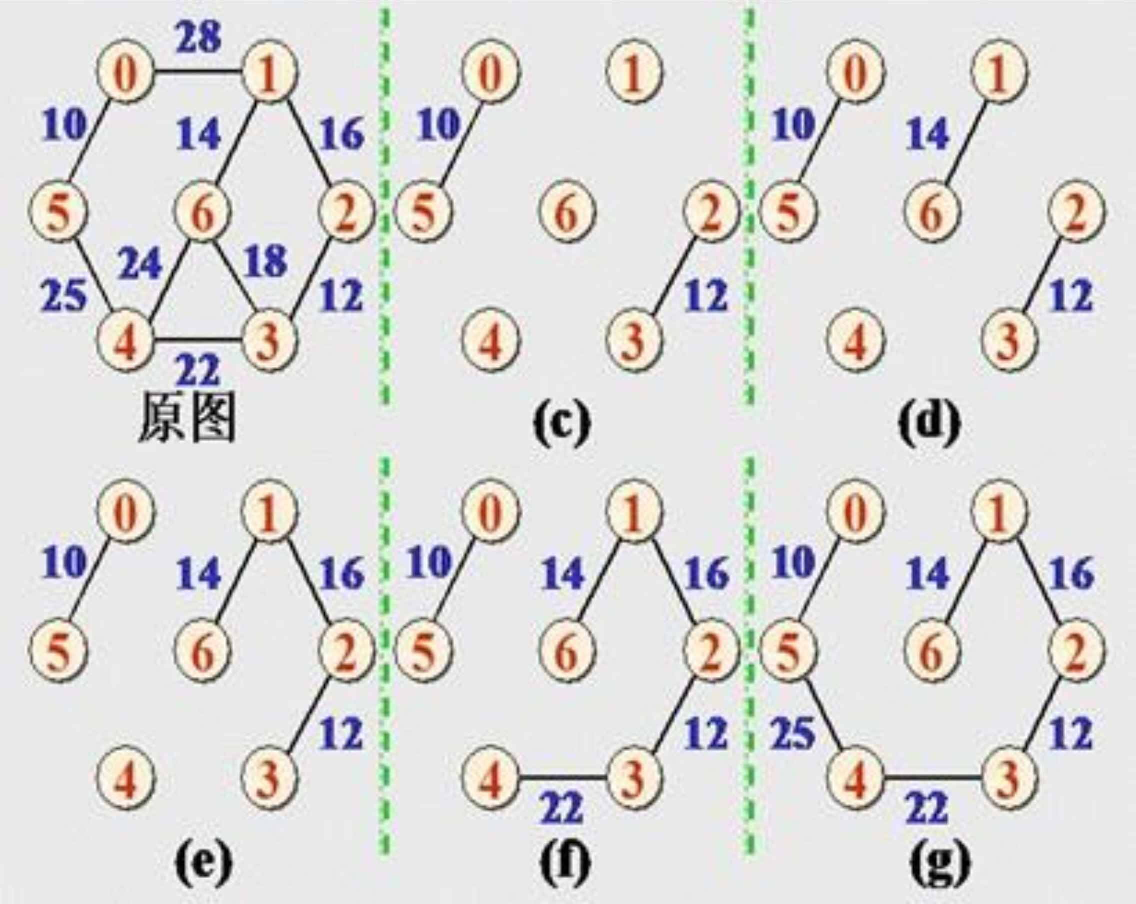
Kruskal's Algorithm Completed (5/5)



Exercise of Prim's Algorithm



Another Example



Homework

- Reading programs:
- 1.grmat.h grlist.h uf.h heap.h
- 2.grkruskl.cpp grkruskm.cpp

Homework

- P410, 11.17
- P411,11.18-11.22

-End-