11.5 Minimal Cost Spanning Trees problem

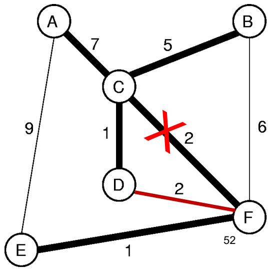
Input: An **undirected**, **connected** graph G = (V,E)

Output: A undirected, connected Acyclic subgraph (Tree) $G_s = (V,E_s)$

1) has minimum total cost as measured by summing the values of all the edges in E_s, and
2) keeps the vertices connected.

MST的特点

1) $|E_s| = |V|-1$， connected
2) G_s is not necessarily unique



52

52

Application example

A minimum spanning tree will provide the optimal solution for road building

53

53

Finding Minimum Spanning Trees

- Prim's algorithm(普里姆算法)
- Kruskal's algorithm(克鲁斯卡尔算法)

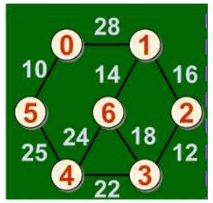
54

54

Prim's algorithm(普里姆算法)

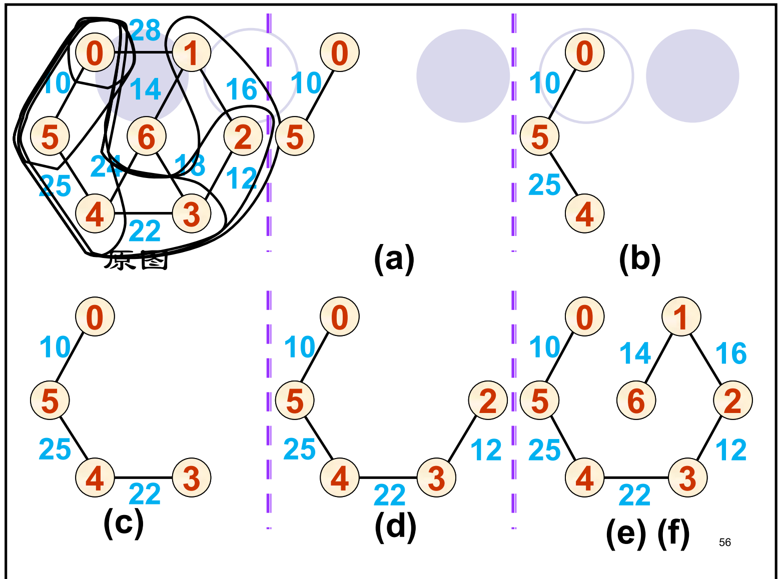
- start with an arbitrary vertex to be the root of a trivial tree
- expand by adding vertices not already in the tree which can be added *most cheaply*

在生成树的构造过程中，|V| 个顶点分属两个集合：已落在生成树上的顶点集 U 和尚未落在生成树上的顶点集V-U，每次从所有连接U中顶点和V-U中顶点的边中选取权值最小的边。



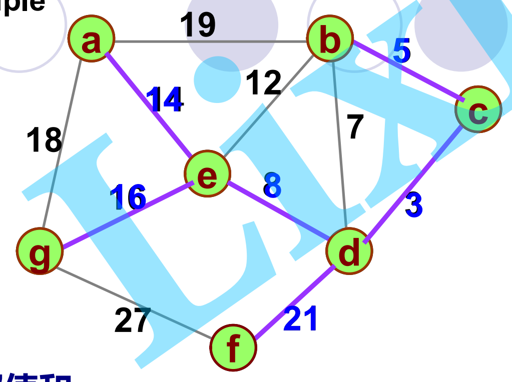
55

55



56

Another Example
Start from a



所得生成树权值和

= 14+8+3+5+16+21 = 67

添边顺序： ae ed dc cb eg df

57

57

Prim's MST Algorithm

```
void Prim( GraphL * G,  int* D,  int s,  Graph* Gmst) {
    int  V[G->n()];  //V 存放U中与各顶点连接边权值最小的顶点，D存放那个最小权值
    int i, w;
    for (i=0; i<G->n(); i++) {    // inital
       V[i]=s;    D[i]=G->weight (s,i);
       G->setMark(i, UNVISITED);    }
    V[s]=-1; G->setMark(s, VISITED);
    for (i=0; i<G->n(); i++) {              // Do vertices
       int  v = minVertex(G, D);  //获取代价最小顶点，即连接U中顶点和V-U中顶点的所有边中权值最小边对应的V-U中顶点
       G->setMark(v, VISITED);
       if (v != s)  Gmst->setEdge(V[v], v, D[v] );
       if (D[v] == INFINITY) return;
       for (w=G->first(v); w<G->n(); w = G->next(v,w))   //update V and D
          if (D[w] > G->weight(v,w)) {
             D[w] = G->weight(v,w);     // Update D
             V[w] = v;      // Update who it came from
          }
    }
}
```

Cost: $O(|V|^2+|E|)$

58

58

Prim's MST Algorithm(continue)

```
int minVertex(G, D) {
   int i, v;
   for (i=0; i<G->n(); i++)   // Set v to an unvisited vertex
      if (G->getMark(i) == UNVISITED)   { v = i; break; }
   for (i++; i<G->n(); i++)    // Now find smallest D value
      if ((G->getMark(i) == UNVISITED) && (D[i] < D[v]))
          v = i;
 return v;
}
```

For minVertex, you can use a heap to acquire an efficient implementation.

59

59

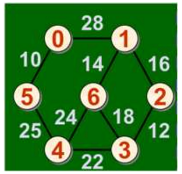
Kruskal算法

基本思路：为使生成树上边的权值之和达到最小，则应使生成树中每一条边的权值尽可能地小。

具体做法：先构造一个只含 n 个顶点的子图 G_S，然后从权值最小的边开始，若它的添加不使 G_S 中产生回路，则在 G_S 中加上这条边，如此重复，直至加上 n-1 条边为止。

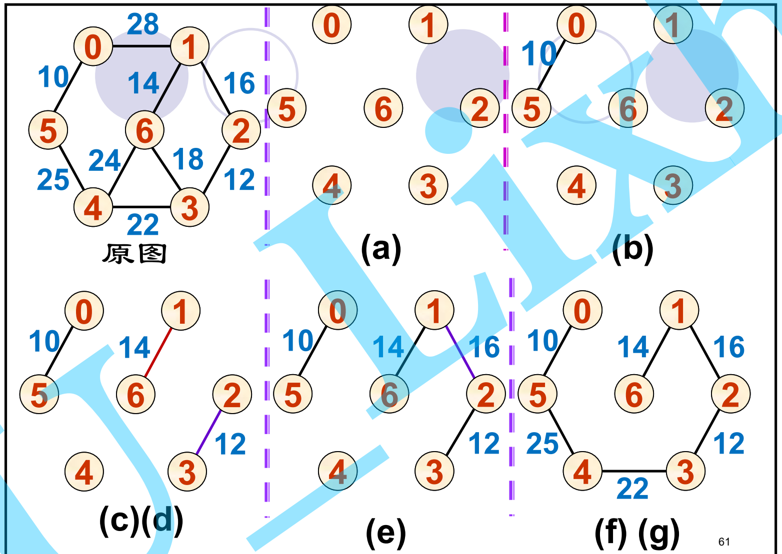
a min-heap

边的2顶点不属于同一棵树（支）



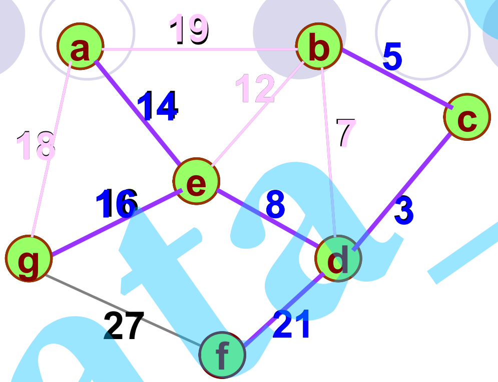
Total cost:
O(|V| + |E| log |E|).

60



61

Another Example



依次选的边为：
cd cb ed ae eg df

所得生成树权值和

= 3+5+8+14+ 16+21 = 67

62

11.5 Shortest Paths Problems

11.5.1 Shortest Paths Problems

11.5.2 Single-Source Shortest Paths

11.5.3 All-Pairs Shortest Paths

63

11.5.1 Shortest Paths Problems(1)

64

64

Shortest Paths Problems(2)

Input: a weighted graph. (Adjacency Matrix or List)

Output: the shortest path (the list of edges) between two vertices and the weighted length of the shortest path(distance of two vertices).

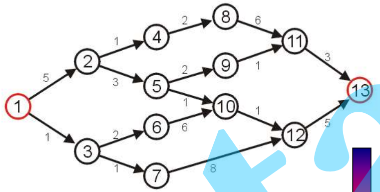
Typical shortest paths problems in a graph:

1. Find shortest path between two named vertices
2. Find shortest path from vertex *s* to all other vertices
   - ✓ Single-Source Shortest Paths
3. Find shortest path between all pairs of vertices

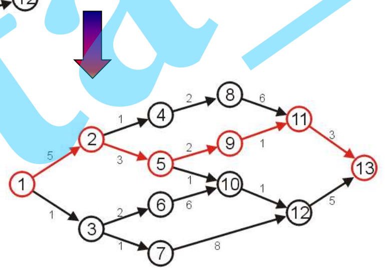
65

65

Shortest path between two named vertices



没有更好的的单独的算法，可作为其他两种情况的副产品



length 14

66

66

11.5.2 Single-Source Shortest Paths

Given start vertex *s*, find the shortest paths from *s* to all other vertices.

Dijkstra's algorithm

Application example:

message broadcast in Computer networks

Two notations:

$d(v_1,v_2)$: the distance from vertex v_1 to v_2, if no path from v_1 to v_2 , $d(v_1,v_2) = \infty$;

$w(v_1,v_2)$: the weight of edge (v_1,v_2), if no edge between v_1 and v_2 ($v1 \neq v2$), $w(v_1,v_2) = \infty$

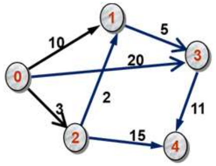
67

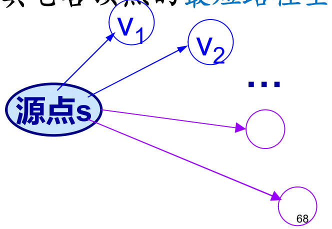
67

单源最短路径— Dijkstra算法思路(1)

按路径长度的递增次序，逐步产生最短路径。

1. 求出长度最短的一条最短路径，
2. 参照(1)已求得的长度最短的最短路径求出长度次短的一条最短路径，
3. 依次类推，直到从顶点v到其它各顶点的最短路径全部求出为止。



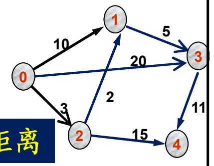


68

单源最短路径— Dijkstra算法(2)

设置数组 Dist和Path，Dist[k] 用来存放从源点s到各顶点 k 的距离 d(s,k)，Path[k]用来追踪到 k的最短路径上的顶点。

1. 初始时，设置 Dist[k] = w(s,k)，Path[k] = s (s,k邻接)或-1(不邻接)，mark[]=0, mark[s]=1, i=1；若不同顶点s与k不邻接，令Dist[k] =∞
2. for (i=1; i<|V|; i++)
   1) 从未标记顶点对应的Dist[k]中选择距离最小的那个顶点，记为v，则v为第i条最短路径的顶点，更新 mark[v] = 1;
   2) 修改与v 邻接且未标记的各顶点对应的*Dist*[*k*]和 Path[k];
      若 *Dist*[*v*]+*w*[*v*][*k*] < *Dist*[*k*]，则
      Dist[*k*] ＝*Dist*[*v*]+w[*v*,*k*]，Path[k] = v;



最终Dist[k]中存放的就是从源s到顶点k的距离

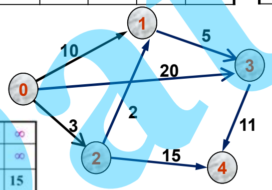
69

Dijkstra's Algorithm Example(1)

Disk[ ]　　Start vertex is 0　　Path [ ]

i \ k	0	1	2	3	4
Initial	0	10	3	20	∞
1	0	5	3	20	18
2	0	5	3	10	18
3	0	5	3	10	18
4-final	0	5	3	10	18

i \ k	0	1	2	3	4
Initial	0	0	0	0	-1
1	0	2	0	0	2
2	0	2	0	1	2
3	0	2	0	1	2
4-final	0	2	0	1	2



0	10	3	20	∞
∞	0	∞	5	∞
∞	2	0	∞	15
∞	∞	∞	0	11
∞	∞	∞	∞	0

邻接矩阵（改造后）

Mark[ ]

i \ k	0	1	2	3	4
Initial	1	0	0	0	0
i = 1	1	0	1	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	1	1	1	1	1

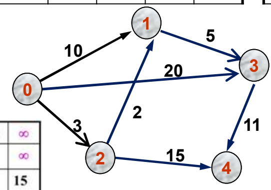
70

Dijkstra's Algorithm Example（2）

Disk[ ]　　Start vertex is 2　　Path [ ]

i \ k	0	1	2	3	4
Initial	∞	2	0	∞	15
1	∞	2	0	7	15
2	∞	2	0	7	15
3	∞	2	0	7	15
4-final	∞	2	0	7	15

i \ k	0	1	2	3	4
Initial	-1	2	2	-1	2
1	-1	2	2	1	2
2	-1	2	2	1	2
3	-1	2	2	1	2
4-final	-1	2	2	1	2



0	10	3	20	∞
∞	0	∞	5	∞
∞	2	0	∞	15
∞	∞	∞	0	11
∞	∞	∞	∞	0

邻接矩阵(改造后）

Mark[]

i \ k	0	1	2	3	4
Initial	0	0	1	0	0
1	0	1	1	0	0
2	0	1	1	1	0
3	0	1	1	1	1
4	1	1	1	1	1

71

Dijkstra's Implementation

```
// shortest path distances from s, return them in D and P
void Dijkstra(Graph* G, int* D, int * P,  int s) {
   int i, k,v, w;
   for (k=0; k<G->n(); k++) {   //初始化 D[], P[], Mark[]
      D[k]= G->weight(s, k);
      if ( (D[k]==0) && (k!=s) )  { P[k] = -1;  D[k]=INFINITY; }
      else   P[k] = s;
      G->setMark(k, UNVISITED);
   }
   G->setMark(s, VISITED);  // D[s]=0;
   for (i=1; i < G->n(); i++) {      // Do vertices
      v = minVertex(G, D);      if (D[v] == INFINITY)   return;
      G->setMark(v, VISITED);
      for (w=G->first(v); w<G->n(); w = G->next(v,w))   //更新 D[], P[]
         if (D[w] > (D[v] + G->weight(v, w)))
            { D[w] = D[v] + G->weight(v, w);  P[w] = v; }
   }
}
```

Cost: $O(|V|^2 + |E|) = O(|V|^2)$.

72

72

Dijkstra's Implementation(con.)

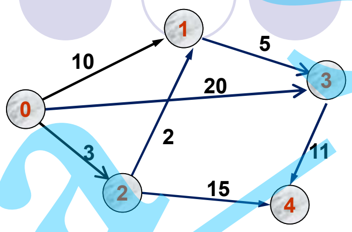
```
int minVertex(Graph* G, int* D) {  // Find min cost vertex
   int i, v;
   for (i=0; i<G->n(); i++)   // Set v to an unvisited vertex
      if (G->getMark(i) == UNVISITED)   { v = i; break; }
   for (i++; i<G->n(); i++)    // Now find smallest D value
      if ((G->getMark(i) == UNVISITED) && (D[i] < D[v]))     v = i;
   return v;
}
void printSPath(Graph* G, int* D, int * P,  int s)  {
 int i,next;
 for ( i=0; i<G->n(); i++)  {
   if (D[i] == INFINITY && i!=s)
        cout << "v"<< i<<"←v"<<s<<": no path"<<endl;
   else  if ( i!=s)  {
        cout<< "v"<<i <<"←";   next = P[i];
        while(next!=s) { cout << "v" << next <<"←";   next= P[next];   }
         cout<< "v"<<s <<":" << D[i] << endl;  }
  }
}
```

73

73

Dijkstra's Algorithm Example(cont.)

```
void main() {
   ……
   Dijkstra(G, D, P, 0);
   printSPath(G,D,P, 0);
}
```



D[ ]

	0	1	2	3	4
final	0	5	3	10	18

P[ ]

	0	1	2	3	4
final	0	2	0	1	2

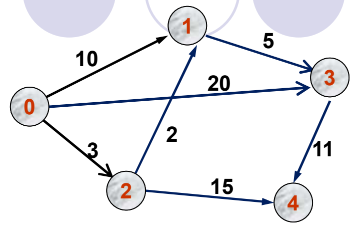
v1←v2←v0 : 5
v2←v0 : 3
v3←v1←v2←v0: 10
v4←v2←v0 : 18

74

74

Dijkstra's Algorithm Example(cont.)

```
void main() {
   ……
   Dijkstra(G, D, P, 2);
   printSPath(G,D,P, 2);
}
```



D[ ]

k	0	1	2	3	4
final	∞	2	0	7	15

P[ ]

k	0	1	2	3	4
final	-1	2	2	1	2

v0←v2: no path
v1←v2 : 2
v3←v1←v2: 7
v4←v2 : 15

75

75

Dijkstra's Algorithm

也可用小堆(优先队列)来实现最短路径的逐步寻找(minVertex), 具体程序见课本p380 Figure11.17

Cost: O((|V| + |E|)log(E))

Graph is sparse, heap-based is better

Cost: $O(|V|^2 + |E|) = O(|V|^2)$.

Graph is dense, using MinVertex is better

11.5.3 All-Pairs Shortest Paths

For every vertex $u, v \in V$, calculate $d(u, v)$.

Method1: Run Dijkstra's Algorithm |V| times.

```
for(i=0; i<G->n(); i++)
  { Dijkstra(G, D, P, i);  printSPath(G,D,P, i); }
```

$O(|V|^3)$

11.5.3 All-Pairs Shortest Paths

For every vertex $u, v \in V$, calculate $d(u, v)$.

Method2: Floyd's Algorithm.

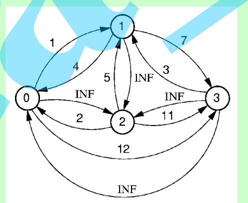
- **Defination: a *k*-path from *u* to *v*: any path whose intermediate vertices have indices less than *k*.**
- **For example**

  1, 3 is a 0-path

  2, 0, 3 is a 1-path

  1, 0, 2, 3 is a 3-path

  all path are 4-path



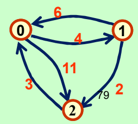
Floyd's Algorithm

Floyd(弗洛伊德)算法：从 v到 w 的所有可能存在的路径中，选出一条长度最短的路径。

- **define $D^k(v,w)$ to be the length of the shortest k-path from v to w.**
- **assume that we already know all shortest k-path from any vertex v to any w**
- **the shortest (k+1)-path from v to w either goes through vertex k or not**
  - **if $D^k(v,w) > D^k(v,k)+D^k(k,w)$, then it go through k, it is the shortest k-path from v to k followed by the shortest k-path from k to w:**

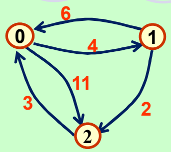
    $D^{k+1}(v,w) = D^k(v,k)+D^k(k,w)$
  - **Otherwise, it is not go through k, it equal to shortest k-path from v to w:**

    $D^{k+1}(v,w)=D^k(v,w)$
- **The final shortest |V|-path from v to w must be the shortest path from v to w。**



Floyd's Algorithm -----an example

$$D^0=\begin{pmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & \infty & 0\end{pmatrix}$$



$$D^1=\begin{pmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & 7 & 0\end{pmatrix}$$

$$D^2=\begin{pmatrix}0 & 4 & 6\\ 6 & 0 & 2\\ 3 & 7 & 0\end{pmatrix}$$

$$D^3=\begin{pmatrix}0 & 4 & 6\\ 5 & 0 & 2\\ 3 & 7 & 0\end{pmatrix}$$

$V_0 \rightarrow V_1$: 4
$V_0 \rightarrow V_1 \rightarrow V_2$: 6
$V_1 \rightarrow V_2 \rightarrow V_0$: 5
$V_1 \rightarrow V_2$: 2
$V_2 \rightarrow V_0$: 3
$V_2 \rightarrow V_0 \rightarrow V_1$: 7

80

80

Floyd's Algorithm implementation

```
void Floyd(Graph * G, int **D, int **P) { all-pairs shortest paths algorithm
  // D[G->n()][G->n()] , P[G->n()][G->n()]  to store distances and path
  int i, j, k;
  for (i=0; i<G->n(); i++) // Initialize                    初始化D,P (D0, P0 )
    for (j=0; j<G->n(); j++) {
       D[i][j] = G->weight(i, j);
       if ( (D[i][j]==0)  && (j!=i) ) { P[i][j] = -1; D[i][j]=INFINITY; }
       else   P[i][j] = j;
    }
  for (k=0; k<G->n(); k++)    // Compute all k paths
   for (i=0; i<G->n(); i++)                 更新D,P (Dk+1, Pk+1) ),k=0,..|V|-1
     for (j=0; j<G->n(); j++)
       if (D[i][j] > (D[i][k] + D[k][j])) {          cost: O(|V|3)
          D[i][j] = D[i][k] + D[k][j];    P[i][j]=P[i][k];
       }
}
```

81

81

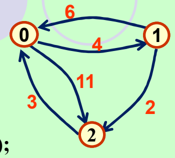
Floyd's Algorithm implementation(con.)

```
void printASPath (Graph * G, int **D, int **P) {
  for (int i=0; i<G->n(); i++) {
    for (int j=0; j<G->n(); j++) {
        if(i ==j)  continue;
        int next = P[i][j];
        if (next == -1)  {
           cout<< "v"<<i<<" --> v"<<j<<":   no path"<<endl;
           continue;
        }
        cout<<"v"<<i;
        while (next!=j)  {  cout << "-->v"<<next;   next=P[next][j];  }
        cout<<"-->v"<<j <<":  "<< D[i][j];
    }
  }
}
```

82

82

Floyd's Algorithm -----an example




```
void main() {
  ......
  Floyd(G, D, P);
  printASPath(G,D,P);
}
```

$$D^0=\begin{pmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & \infty & 0\end{pmatrix} \quad P^0=\begin{pmatrix}0 & 1 & 2\\ 0 & 1 & 2\\ 0 & -1 & 2\end{pmatrix}$$

$$D^1=\begin{pmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & 7 & 0\end{pmatrix} \quad P^1=\begin{pmatrix}0 & 1 & 2\\ 0 & 1 & 2\\ 0 & 0 & 2\end{pmatrix}$$

$$D^2=\begin{pmatrix}0 & 4 & 6\\ 6 & 0 & 2\\ 3 & 7 & 0\end{pmatrix} \quad P^2=\begin{pmatrix}0 & 1 & 1\\ 0 & 1 & 2\\ 0 & 0 & 2\end{pmatrix}$$

$$D^3=\begin{pmatrix}0 & 4 & 6\\ 5 & 0 & 2\\ 3 & 7 & 0\end{pmatrix} \quad P^3=\begin{pmatrix}0 & 1 & 1\\ 2 & 1 & 2\\ 0 & 0 & 2\end{pmatrix}$$

$V_0 \rightarrow V_1$: 4
$V_0 \rightarrow V_1 \rightarrow V_2$: 6
$V_1 \rightarrow V_2 \rightarrow V_0$: 5
$V_1 \rightarrow V_2$: 2
$V_2 \rightarrow V_0$: 3
$V_2 \rightarrow V_0 \rightarrow V_1$: 7

83

83

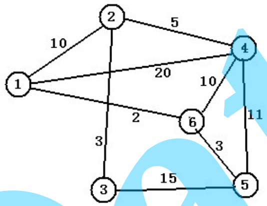
本章我们学到了

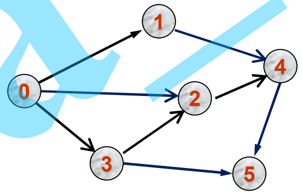
- 图的概念、术语及描述
- 图的遍历
  - DFS and BFS
- 拓扑排序， DAG
  - based on DFS
  - based on BFS
- 最小生成树：连通无向加权图 → 最小权值树
  - Prim’s algorithm
  - Kruskal’s algorithm
- 最短路径问题
  - 单源问题， Dijkstra’s Algorithm
  - 任意两顶点之间的最短路径，Floyd’s Algorithm

All End

课后练习

1. Show the shortest paths from Vertex 4 to all other vertices by using Dijkstra’s shortest-paths algorithm on the following left graph. Show the values of Dist[] as each vertex is processed.

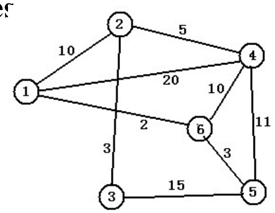




2. Give out the topologic sort result of the above right graph using BFS based method. Be sure to display the Queue after each Enqueque and Enqueue.

课后练习

1）List the order in which the edges of the given graph are added when running Kruskal’s MST algorithm.



2）Show the final MST and compute its cost