数据结构与算法分析

华中科技大学软件学院

2017年秋

大纲

1. 图的表示
2. 拓扑排序与结点访问
3. 最短路径
4. 最小生成树

课程计划

- 已经学习了
  - 排序算法的重要性
  - 比较交换相邻元素的排序
  - 基于比较的最优排序
  - 排序算法的分析

课程计划

- 已经学习了
  - 排序算法的重要性
  - 比较交换相邻元素的排序
  - 基于比较的最优排序
  - 排序算法的分析
- 即将学习算法设计思想
  - 图的表示
  - 图的结点访问
  - 最小生成树
  - 最短路径

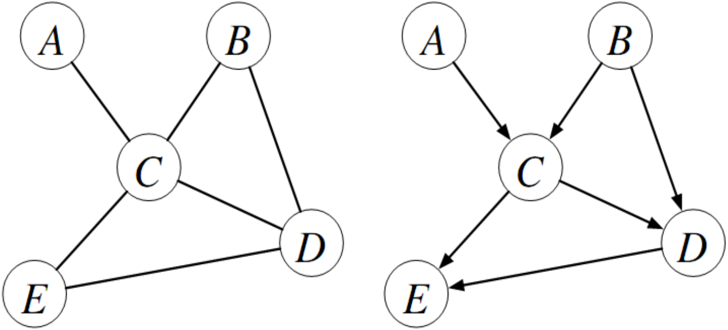
Roadmap

1. 图的表示
2. 拓扑排序与结点访问
3. 最短路径
4. 最小生成树

Graph Theory

- Graph G = (V, E)： $V = \{v_i : 1 \le i \le n\}$，顶点集（节点），E 是 $V \times V$的子集 = 边集（弧）
- 可以用图来表示任何关系：每个节点都是一项，若两项相关，则2节点之间有一条边
  - 有向图Directed graph（“digraph”），边有方向
  - 正则图Regular graph（“bi-directional”），无向
- 每条边可以有“权”，例如，两点之间的距离/值

Graphs

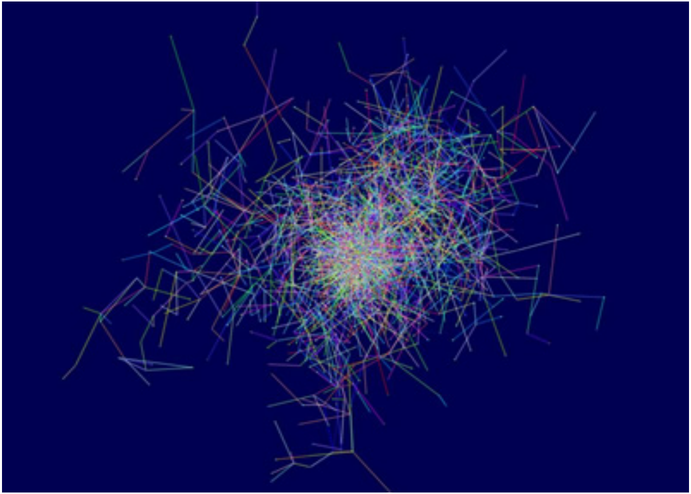


图的应用

- 从A到B花费成本最少的路径是(权的和最少)? 从A到B的最短路径是?(边数最少)? 应在哪里添加直达航线?
- 其他应用:
  - 模拟地面交通:
  - 瓶颈路段在哪里?
  - 神经网络
  - 马尔可夫链
  - 网络图

Erdös Collaboration Graph

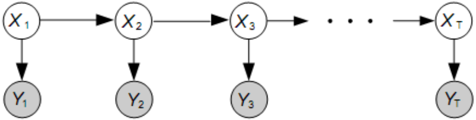
Erdös' 第二邻域协作图的随机子图



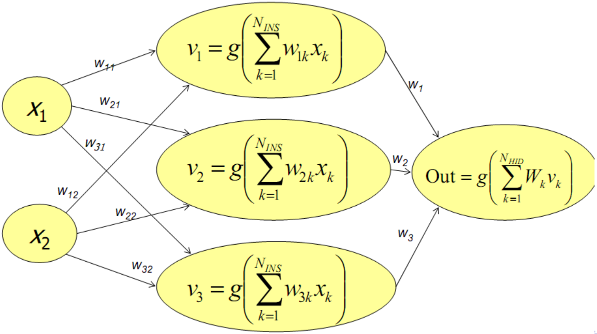
Erdös Numbers

- 因此，Erdös的中位数为5，均值为4.65，标准差为1.21
  - Erdös number 1 – 504 people
  - Erdös number 2 – 6593 people
  - Erdös number 3 – 33605 people
  - Erdös number 4 – 83642 people
  - Erdös number 5 – 87760 people
  - Erdös number 6 – 40014 people
  - Erdös number 7 – 11591 people
  - Erdös number 8 – 3146 people
  - Erdös number 9 – 819 people
  - Erdös number 10 – 244 people
  - Erdös number 11 – 68 people
  - Erdös number 12 – 23 people
  - Erdös number 13 – 5 people

- 隐马尔可夫模型

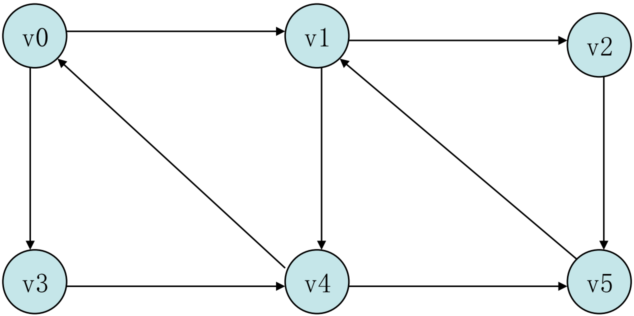


- 人工神经网络

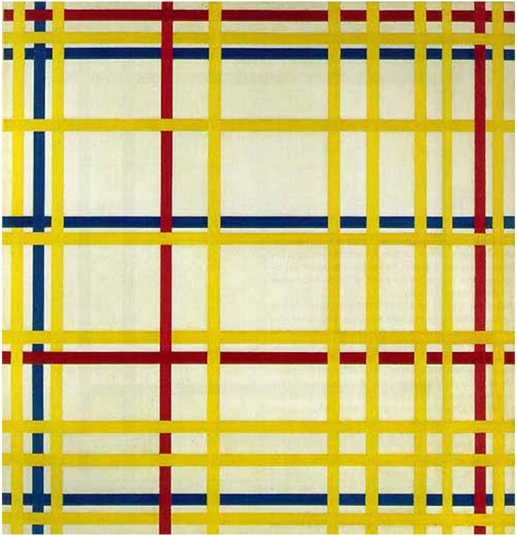


邻接矩阵

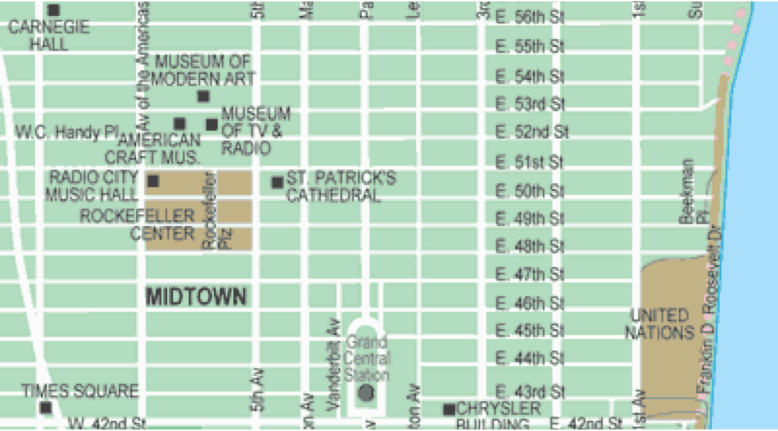
自然的方式：邻接矩阵，$|V| \times |V|$ 矩阵 A[v1][v2] = 1（或放入边的值）当且仅当 v_1, v_2 相邻



Representation of Graphs



Representation of Graphs



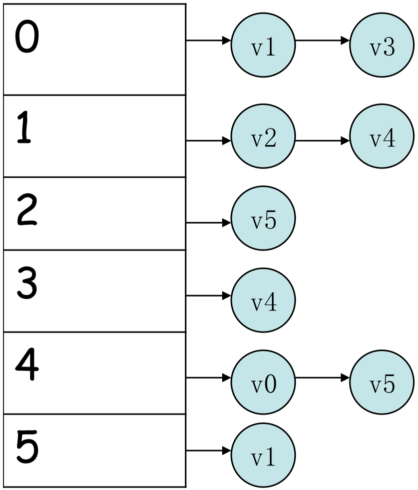
稠密/稀疏的矩阵

- 若$|E| = \Theta(|V|^2)$, 则图是稠密的，在(几乎)每一对结点之间都有边
- 为曼哈顿的每个十字路口创建节点
  - 为每个 连接两交叉口的 街道单元，创建边
  - 假定有 3000 个 4-way 交叉点，2入，2出 $\rightarrow$ 2*3000 = 6000 条边，3000^2 = 9,000,000 个 相邻矩阵中的项

邻接表

- 对于稀疏图，最好用另一种方式实现
- 对每个节点：创建相邻节点的链接列表
- 对于有向图，每条边有1个入口。
- 对于正则图，每条边有2个入口
- 无论哪种方式都有：O(|E| + |V|)

邻接表



Which is better?

Problems	邻接矩阵	邻接表
Adj(x, y)?	O(1)	deg(x) or deg(y)
Find deg(x)	$\|V\|$	deg(x)
Sparse	$\|V\|^2$	$\|V\|+\|E\|$
Dense	$\|V\|^2$	$\|V\|^2$
Add/del edge	O(1)	$\|V\|$
Traverse graph	$\|V\|^2$	$\|V\|+\|E\|$

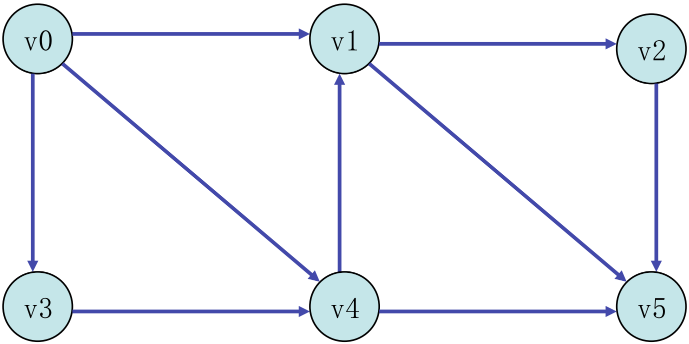
通常认为，邻接表是更好的

Roadmap

Topological sort

- 对于给定的有向无圈图，可提问：我们用什么顺序访问节点？
- 顶部排序：节点的列表。如果存在一条从v1到v2的路径，那么v1出现在v2之后。
  - 一般来说，图的可能的顶部排序有很多种。
  - 如果图有圈，则没有很好的定义(拓扑排序不可能)。
- 谁先被访问？
  - 例：每门课程都有先决条件，查找所有合法的课程顺序
  - 重要的应用：任务管理

Example



Top-sort

```
void topSort (Graph G)
{
    int ctr;
    Vertex v, w;

    for (ctr = 0; ctr < NUM_VERTS; ctrl++)
    {
        v = findInDeg0Vert();
        if (v == NULL)
        {
            printf ("A␣cycle␣found\n");
            break;
        }

        topNum[v] = ctr;
        for each w adjacent to v
            indegree[w]--;
    }
}
```

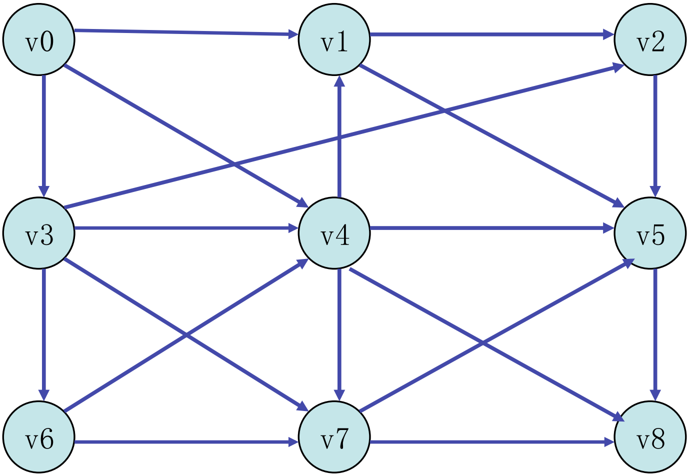
Top Sort 运算法则

- findInDeg0Vert() 子程序
  - 遍历顶点的数组
  - 每次调用花费O(|V|)时间 $\rightarrow O(|V|^2)$
  - 对于稠密图来说，还行
- 更优解：把所有入度为0的顶点放在一个特殊的盒子中
  - 每次减少一个结点的入度,若当前值是0.则把它放入盒子
  - 盒子的形式是 堆栈(stack) 或 队列(queue)

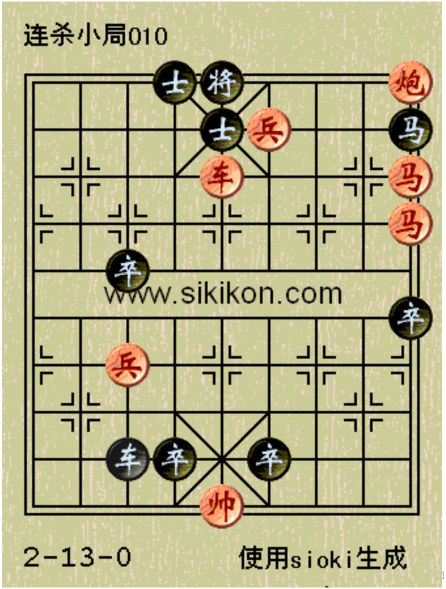
Top-sort with Queue

```
 void topsort()
{
    Queue q;
    int ctr = 0;
    Vertex v,w;

    q = createQueue (NumVertex); MakeEmpty (Q);
    for each vert v
        if (indegree[v] == 0)
            enqueue (v, q);

    while (!Isempty (q))
    {
        v = dequeue (q);
        topNum[v] = ++ctr;
        for each w adj to v
            if (--indegree[w] == 0)
                enqueue (w, q);
    }

    if (ctr != NUM_VERTS)
        printf ("A␣cycle␣is␣found\n");
    disposeQueue (q);
}
```

Exercise



Visiting Graph Nodes



搜索算法

- BFS：宽度优先搜索 = 层序遍历
- DFS：深度优先搜索 = 先序遍历

DFS

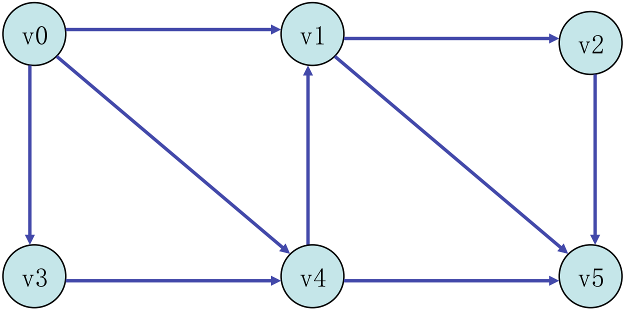
```
void dfs(vert v)
{
    visited[v] = TRUE;

     vw
        if (!visited[w])
             dfs(w);
}
```

BFS

```
void bfs (vert v)
{
    queue Q;
    vert w;

    makeEmptyQueue (Q) ;
    visited[v] = TRUE;
    enqueue (Q, v) ;

    while (!isEmpty (Q ))
    {
        v = dequeue (Q);
        vw
            if (!visited [w])
            {
                visited (w) = TRUE;
                enqueue (Q, w);
            }
    }

    disposeQueue (Q);
}
```

Starting from V0

Try DFS and BFS



Roadmap

1. 图的表示
2. 拓扑排序与结点访问
3. **最短路径**
4. 最小生成树

Shortest-path Problems

- 单源最短路径问题： 给定一个加权图G和一个顶点s 找到从s到所有节点的最短路径
- 从未加权的图开始
- 做的广度优先搜索

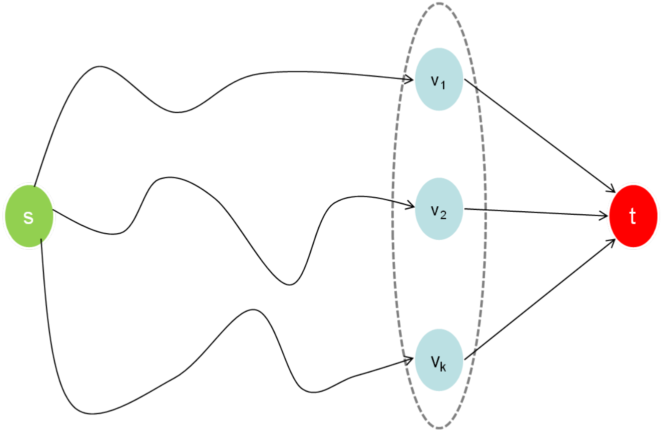
Shortest Path

Theorem

一条最短路径的子路径是一条最短路径

- 三角不等式
- $d(s,t) = \min_{(v,t)\in E}(d(s,v) + w(v,t))$
- Bellman Ford算法适用于负权值图，O(|E||V|)

$$d(s,t) = \min_{(v,t)\in E}(d(s,v) + w(v,t))$$



Stortest Path

```
for v != s
    initialize d[v][0] = INFTY;

for all i
    d[t][i]=0;

for i=1 to n-1
    for each v != s
        d[v][i] = min ((v,x) in E (len(v,x)
         + d[x][i-1]))

for each v
    output d[v][n-1].
```

Unweighted Algorithm Code

```
void unweighted (Vertex s)
{
    int currDist;
    Vertex v, w;
    dist[s] = 0;

    for (currDist = 0; currDist < NUM_VERTS; currDist++)
        for each vert v
            if (!known[v] && dist[v] == currDist)
            {
                known[v] = TRUE;
                for each w adjacent to v
                    if (dist[w] == INFTY)
                    {
                        dist[w] = currDist+1;
                        path[w] = v;
                    }
             }
}
```

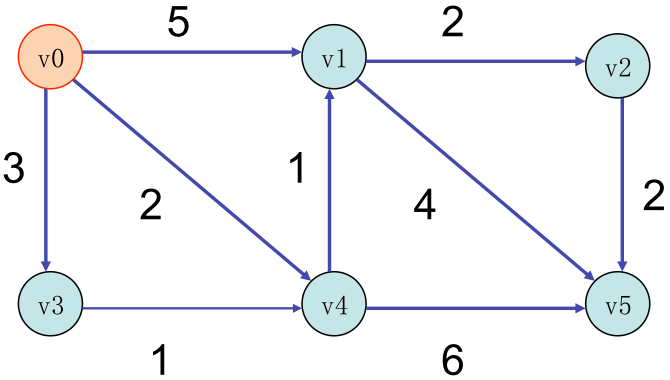
With the help of Queue

```
void unweighted (Vertex s)
{
    Queue q;
    Vwrtex v, w;

    createQueue (q); makeEmpty (q);
    enqueue (s, q);

    while (!isEmpty (q))
    {
        v = dequeue (q);
        known[v] = TRUE;

        for each w adjacent to v
            if (dist[w] == INF)
            {
                dist[w] = dist[v] + 1;
                path[w] = v;
                enqueue (w);
            }
    }
    disposeQueue (q);
}
```

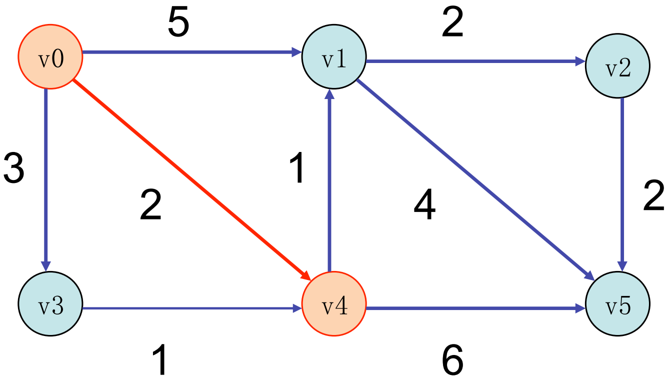
Dijkstra's Algorithm

- 加权最短路径为正权值
- 复杂度 $O(|E|\log|V|)$
- 贪婪算法:总是选择最短的边
- 想法:在每次迭代中，选择最小距离的未知节点
- 每个节点有3条信息
  - 已知的布尔值，是否确定了最短的距离
  - d_v 到目前为止,最短的距离
  - p_v 之前的节点

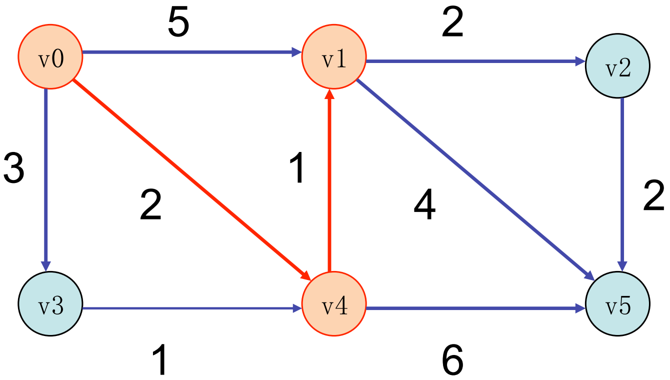
Example



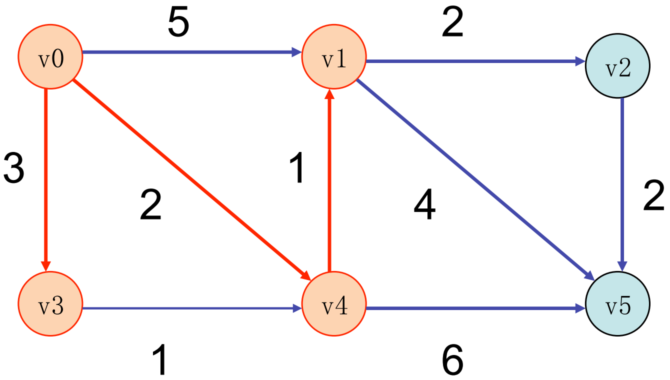
Example



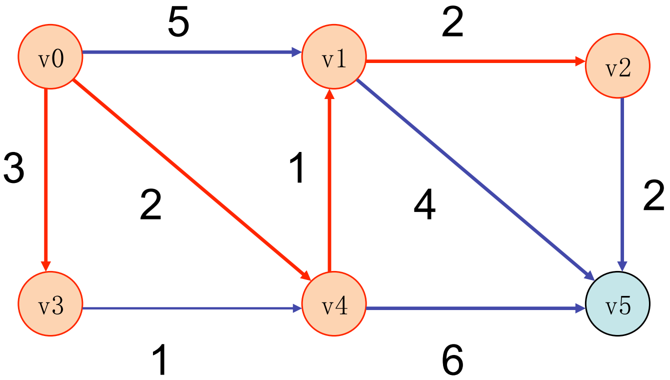
Example



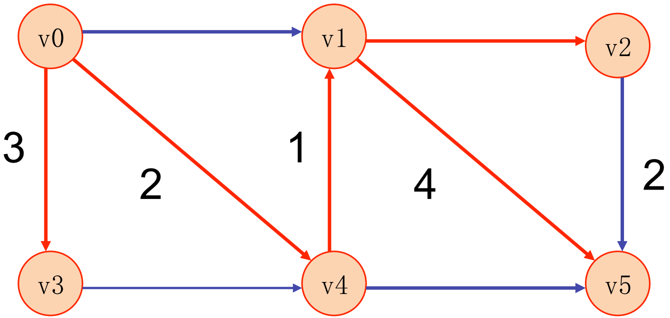
Example



Example



Example



Dijkstra Code

```
typedef struct TableEntry
{
    List header;
    boolean known;
    DistType dist;
    Vertex path;
}

void initTable (Veterx s, Graph G, Table T)
{
    int i;

    readGraph (G, T);
    for (i = 0; i < NUM_VERTS; i++)
    {
        T[i].known = FALSE;
        T[i].dist = INF;
        T[i].path = NULL;
    }
    T[s].dist = 0;
}
```

Dijkstra Code

```
void printPath (Vertex v, Table T)
{
    Vertex v, w;

    if (T[v].path != NotAVertex)
    {
        printPath (T[v].path, T);
        printf ("␣to␣");
    }

    printf ("%d", v);
}
```

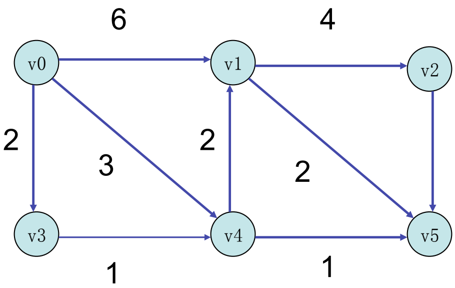
Dijkstra Code

```
void dijkstra (Vertex s, Table T)
{
    Vertex v, w;
    T[s].dist = 0;

    while (TRUE)
    {
        v = smallest-dist unknown vertex;
        if (v == NotAVertex) break;
        T[v].known = TRUE;

        for each w adjacent to v
            if (!T[w].known && T[v].dist + Cvw < T[w].dist)
            {
                T[w].dist = T[v].dist + Cvw;
                T[w].path = v;
            }
      }
}
```

Dijkstra's Complexity

- 怎样找到到未知顶点的最小距离？
- 如果做线下查找，每个消耗 $\Theta(|V|)$ 时间 总共消耗 $\Theta(|E|+|V|^2)=\Theta(|V|^2)$
- 对节点较多的图好用，图如果比较稀疏，则效果不好
- 更好的方法：把未知节点放在 minQueue Select v. known == true $\rightarrow$ delMin，每个的复杂度为 $\log|V|$
- 如果更改 dist[w] 如何?
- 成为一个 decreaseKey 操作
- 假设有查找元素的方法，或者存储位置 每个的复杂度为 $\log|V|$ ，Total: $\Theta(|E|\log|V|+|V|\log|V|)=\Theta(|E|\log|V|)$

Exercise



基准情况： dist（V0， V0）= 0
递归步骤： dist（V0， Vi）= min（dist（V0， Vi），
dist（V0， Vj）+ Cji），for all edges j->i

Initial Values

	Known	path	dist	0
V0	True	Null	0	0
V1	False	Null		INF
V2	False	Null		INF
V3	False	Null		INF
V4	False	Null		INF
V5	False	Null		INF

Round 1

	Known	path	dist	0	1
V0	True	Null	0	0	0
V1	False	V0		INF	6
V2	False	Null		INF	INF
V3	False	V0	2	INF	2
V4	False	V0		INF	3
V5	False	Null		INF	INF

Round 1

	Known	path	dist	0	1
V0	True	Null	0	0	0
V1	False	V0		INF	6
V2	False	Null		INF	INF
V3	True	V0	2	INF	2
V4	False	V0		INF	3
V5	False	Null		INF	INF

Round 2

	Known	path	dist	0	1	2
V0	True	Null	0	0	0	0
V1	False	V0		INF	6	6
V2	False	Null		INF	INF	INF
V3	True	V0	2	INF	2	2
V4	False	V0	3	INF	3	3
V5	False	Null		INF	INF	INF

Round 2

	Known	path	dist	0	1	2
V0	True	Null	0	0	0	0
V1	False	V0		INF	6	6
V2	False	Null		INF	INF	INF
V3	True	V0	2	INF	2	2
V4	True	V0	3	INF	3	3
V5	False	Null		INF	INF	INF

Round 3

	Known	path	dist	0	1	2	3
V0	True	Null	0	0	0	0	0
V1	False	V0		INF	6	6	5
V2	False	Null		INF	INF	INF	INF
V3	True	V0	2	INF	2	2	2
V4	True	V0	3	INF	3	3	3
V5	False	V4	4	INF	INF	INF	4

Round 3

	Known	path	dist	0	1	2	3
V0	True	Null	0	0	0	0	0
V1	False	V0		INF	6	6	5
V2	False	Null		INF	INF	INF	INF
V3	True	V0	2	INF	2	2	2
V4	True	V0	3	INF	3	3	3
V5	True	V4	4	INF	INF	INF	4

Round 4

	Known	path	dist	0	1	2	3	4
V0	True	Null	0	0	0	0	0	0
V1	False	V4	5	INF	6	6	5	5
V2	True	Null		INF	INF	INF	INF	INF
V3	True	V0	2	INF	2	2	2	2
V4	True	V0	3	INF	3	3	3	3
V5	True	V4	4	INF	INF	INF	4	4

Round 4

	Known	path	dist	0	1	2	3	4
V0	True	Null	0	0	0	0	0	0
V1	True	V4	5	INF	6	6	5	5
V2	True	Null		INF	INF	INF	INF	INF
V3	True	V0	2	INF	2	2	2	2
V4	True	V0	3	INF	3	3	3	3
V5	True	V4	4	INF	INF	INF	4	4

Final Round

	Known	path	dist	0	1	2	3	4	5
V0	True	Null	0	0	0	0	0	0	0
V1	True	V4	5	INF	6	6	5	5	5
V2	False	V1	9	INF	INF	INF	INF	INF	9
V3	True	V0	2	INF	2	2	2	2	2
V4	True	V0	3	INF	3	3	3	3	3
V5	True	V4	4	INF	INF	INF	4	4	4

Finally

	Known	path	dist	0	1	2	3	4	5
V0	True	Null	0	0	0	0	0	0	0
V1	True	V4	5	INF	6	6	5	5	5
V2	True	V1	9	INF	INF	INF	INF	INF	9
V3	True	V0	2	INF	2	2	2	2	2
V4	True	V0	3	INF	3	3	3	3	3
V5	True	V4	4	INF	INF	INF	4	4	4

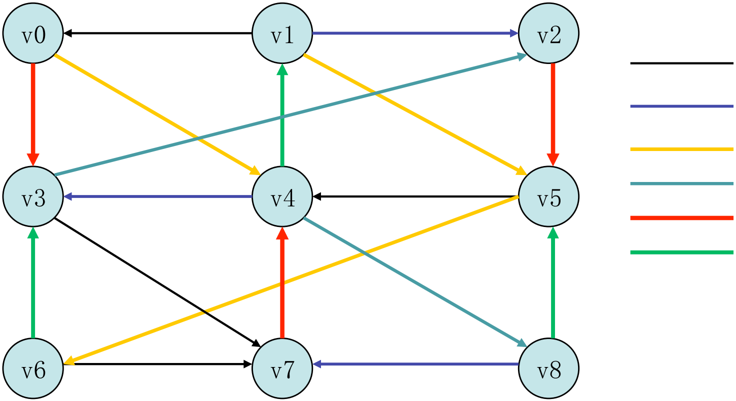
Weighted Negative

```
void weightedNegative (Vertex s, Table T)
{
    Queue = q;
    Vertex v, w;
    q = createQueue (NUM_VERTS); makeEmpty (q);
    enqueue (s q);

    while (!isEmpty (q))
    {
        v = dequeue (q);
        if have seen v |V|+1 times, break;
        for each w adjacent to v
            if (T[v].dist + Cvw < T[w].dist)
            {
                T[w].dist = T[v].dist + cvw;
                T[w].path = v;
                if (! contains(q, w))
                    enqueue (w, q);
            }
    }

    disposeQueue (q);
}
```

Acyclic Graphs

- 如果图是非循环的，Dijkstra更容易
- 改变已知顶点的顺序
- 在一次递归中选择拓扑顺序的顶点
- 如果选择了v,其距离dv不能更低
- 由拓扑次序规则可知，没有未知节点指向它
- 常量选择时间 $\rightarrow$ 没有优先度 Q $\rightarrow \Theta(|E| + |V|)$

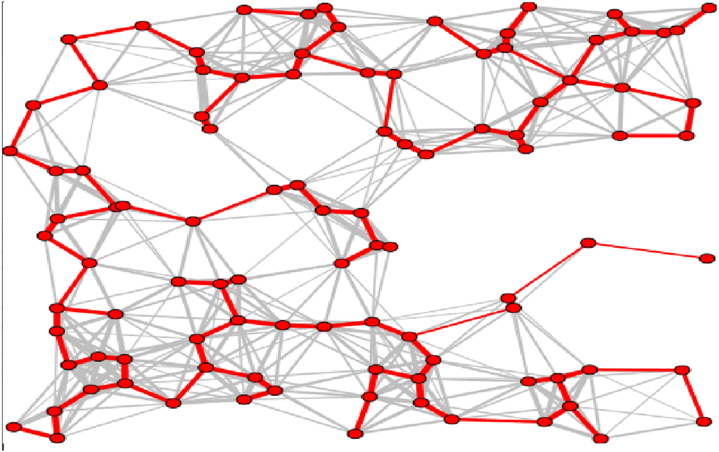
Example



Roadmap

Minimum Spanning Tree

以最低的成本连接所有节点



Greedy Algorithm

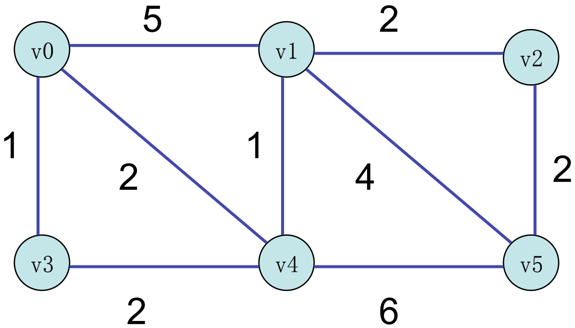
- 想法:给定一个（加权）图，生成包含所有权和最小的节点的树
- 比如： 考虑到在不同房间使用电视机，找到用最少的电缆总长度连接所有电视的方法。
- MST中有多少条边? |V| −1
- 贪婪算法： 创建生成树的边，每增加的一条边都是增加最小代的价边（避免循环）

- Prim’s 算法：创建单个 MST
- Kruskal’s 算法：创建一个连接的MSTs森林

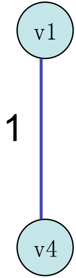
Prim's Algorithm

- 和 Dijkstra's 算法相似，除了dv是连接已知顶点的最短边的重量
- 更新规则： 对于每个未知的顶点w与v相邻，$d_w = \min(d_w, c_{wv})$
- 时间： $O(|V|^2)$ 没有堆， O(|E|log|V|） 有堆

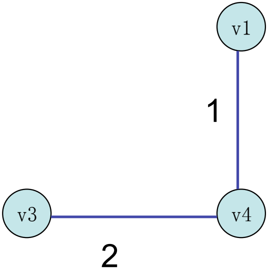
Prim's Algorithm



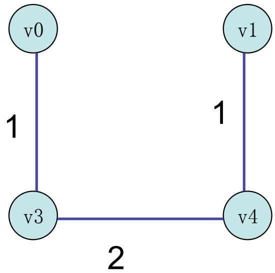
Prim’s Algorithm



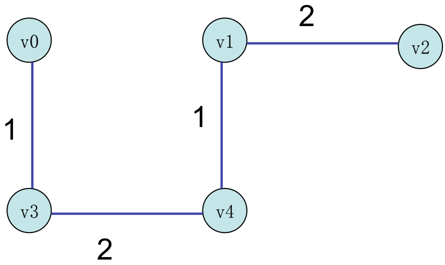
Prim’s Algorithm



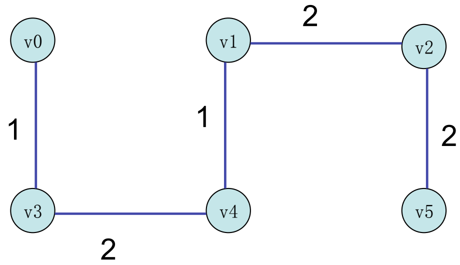
Prim's Algorithm



Prim's Algorithm



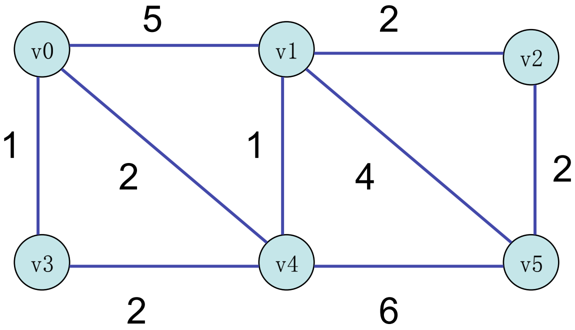
Prim’s Algorithm



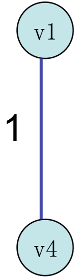
Kruskal's Algorithm

- 选择权重最小的边. 如果它不引起循环，则添加到图中 重复直到已经添加 $|V|-1$ 条边
- 如何选择最小边？可以排序，但是 $|E|\log|E|$. 更好：建立时间为$|E|$的边优先队列Q，然后提取最小边
- 如何判断是否循环? 将所有连接的边放在一个集合中， 集合中两个边的两头 $\rightarrow$ 可以构成循环
- 时间：$O(|E|\log|E|) = O(|E|\log|V|)$. 实践中，Kruskal's 比 Prim's 快

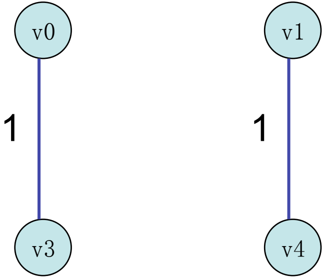
Kruskal’s Algorithm



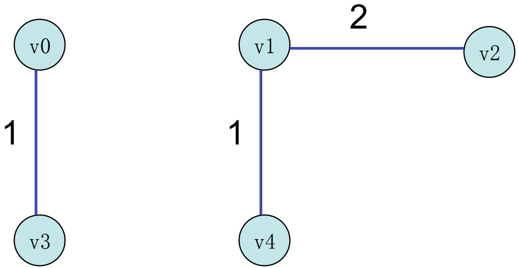
Kruskal’s Algorithm



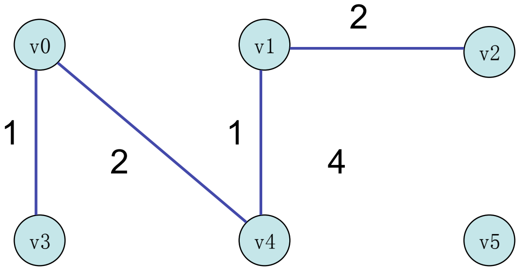
Kruskal’s Algorithm



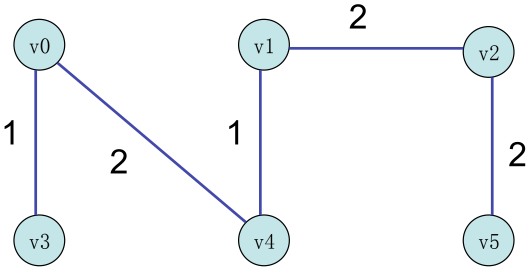
Kruskal’s Algorithm



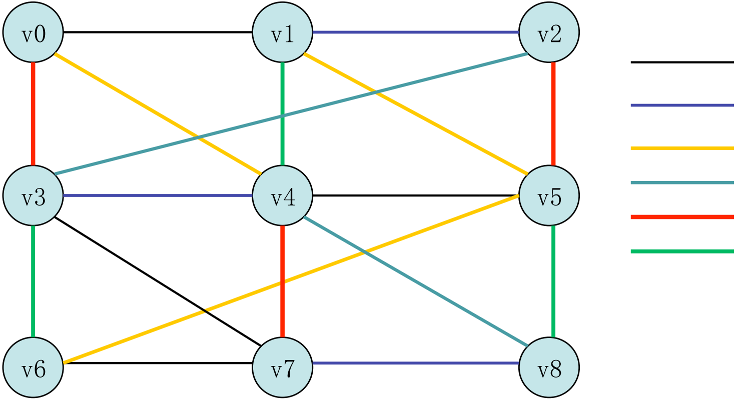
Kruskal's Algorithm



Kruskal's Algorithm



Exercise



小结

- 图可以用邻接矩阵或邻接链表表示
- 图的结点可以通过深度优先或广度优先的算法实现
- 拓扑排序将有向无环图中的偏序关系转换为线性关系
- 贪婪算法：Dijkstra算法计算最短路径，Prim和Kruskal算法寻找最小生成树