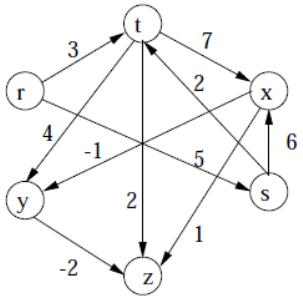
在有向无环图中求最长路径

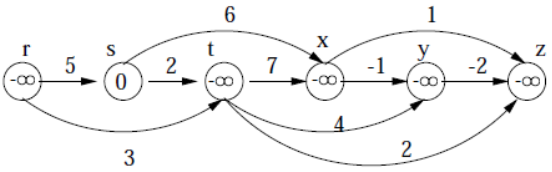
给定一个带权有向无环图及源点S,在图中找出从S出发到图中其它所有顶点的最长距离。

对于一般的图，求最长路径并不向最短路径那样容易，因为最长路径并没有最优子结构的属性。实际上求最长路径属于NP-Hard问题。然而，对于有向无
环图，最长路径问题有线性时间的解。思路与通过使用拓扑排序在线性时间求最短路径[1]一样。

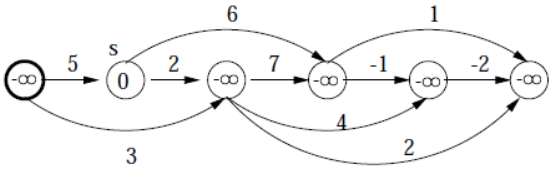
首先初始化到所有顶点的距离为负无穷大，到源点的距离为0，然后找出拓扑序。图的拓扑排序代表一个图的线性顺序。（图b是图a的一个线性表示）。
当找到拓扑序后，逐个处理拓扑序中的所有顶点。对于每个被处理的顶点，通过使用当前顶点来更新到它的邻接点的距离。



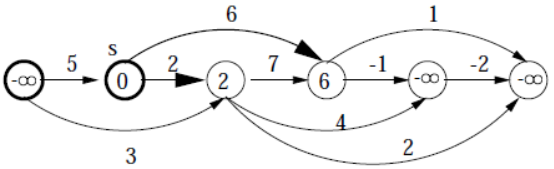
(a)



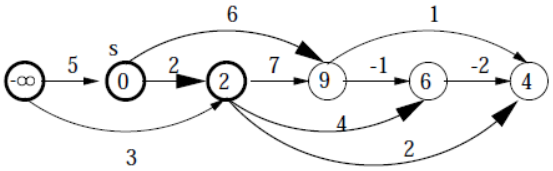
(b)



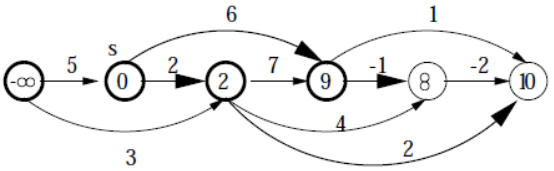
(c)



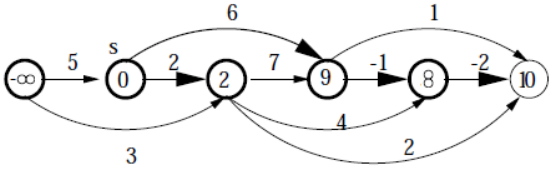
(d)



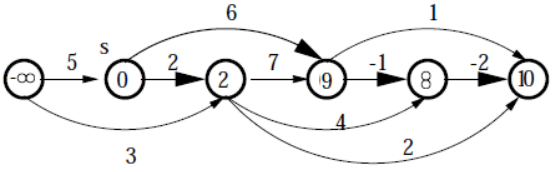
(e)



(f)



(g)



(h)

图(b)中，到点s的距离初始化为0，到其它点的距离初始化为负无穷大，而图(b)中的边表示图(a)中边的权值。
图(c)中，求得从s到r的距离为负无穷。
图(d)中，求得s到t的最长距离为2，到x的最长距离为6。
图(e)至图(h)依次求得可达点间的最长距离。

下面是寻找最长路径的算法

1. 初始化 dist[] = {NINF, NINF, ….} ， dist[s] = 0 。s是源点，NINF表示负无穷。dist表示源点到其它点的最长距离。
2. 建立所有顶点的拓扑序列。
3. 对拓扑序列中的每个顶点u执行下面算法。
   对u的每个邻接点v
   if (dist[v] < dist[u] + weight(u, v)) ……dist[v] = dist[u] + weight(u, v)

下面是C++的实现。

```
1. // A C++ program to find single source longest distances in a DAG
2. #include <iostream>
3. #include <list>
4. #include <stack>
5. #include <limits.h>
6. #define NINF INT_MIN
7. using namespace std;
8.
9. //图通过邻接表来描述。邻接表中的每个顶点包含所连接的顶点的数据，以及边的权值。
10. class AdjListNode
11. {
12.     int v;
13.     int weight;
14. public:
15.     AdjListNode(int _v, int _w)  { v = _v;  weight = _w;}
16.     int getV()       {  return v;  }
```

```
    int getWeight()  {  return weight; }
};

// Class to represent a graph using adjacency list representation
class Graph
{
    int V;    // No. of vertices'

    // Pointer to an array containing adjacency lists
    list<AdjListNode> *adj;

    // A function used by longestPath
    void topologicalSortUtil(int v, bool visited[], stack<int>
&Stack);
public:
    Graph(int V);   // Constructor

    // function to add an edge to graph
    void addEdge(int u, int v, int weight);

    // Finds longest distances from given source vertex
    void longestPath(int s);
};

Graph::Graph(int V) // Constructor
{
    this->V = V;
    adj = new list<AdjListNode>[V];
}

```

```
46. void Graph::addEdge(int u, int v, int weight)
47. {
48.     AdjListNode node(v, weight);
49.     adj[u].push_back(node); // Add v to u’s list
50. }
51.
52. // 通过递归求出拓扑序列. 详细描述，可参考下面的链接。
53. // http://www.geeksforgeeks.org/topological-sorting/
54. void Graph::topologicalSortUtil(int v, bool visited[], stack<int>
&Stack)
55. {
56.     // 标记当前顶点为已访问
57.     visited[v] = true;
58.
59.     // 对所有邻接点执行递归调用
60.     list<AdjListNode>::iterator i;
61.     for (i = adj[v].begin(); i != adj[v].end(); ++i)
62.     {
63.         AdjListNode node = *i;
64.         if (!visited[node.getV()])
65.             topologicalSortUtil(node.getV(), visited, Stack);
66.     }
67.
68.     // 当某个点没有邻接点时，递归结束，将该点存入栈中。
69.     Stack.push(v);
70. }

1. // 根据传入的顶点，求出到到其它点的最长路径. longestPath使用了
2. // topologicalSortUtil() 方法获得顶点的拓扑序。
3. void Graph::longestPath(int s)
4. {
```

```
    stack<int> Stack;
    int dist[V];

    // 标记所有的顶点为未访问
    bool *visited = new bool[V];
     for (int i = 0; i < V; i++)
        visited[i] = false;

    // 对每个顶点调用topologicalSortUtil，最终求出图的拓扑序列存入到Stack中。
    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            topologicalSortUtil(i, visited, Stack);

    //初始化到所有顶点的距离为负无穷
    //到源点的距离为0
    for (int i = 0; i < V; i++)
        dist[i] = NINF;
    dist[s] = 0;

    // 处理拓扑序列中的点
    while (Stack.empty() == false)
    {
        //取出拓扑序列中的第一个点
        int u = Stack.top();
        Stack.pop();

        // 更新到所有邻接点的距离
        list<AdjListNode>::iterator i;
        if (dist[u] != NINF)
```

```
        {
          for (i = adj[u].begin(); i != adj[u].end(); ++i)
            if (dist[i->getV()] < dist[u] + i->getWeight())
              dist[i->getV()] = dist[u] + i->getWeight();
        }
    }

    // 打印最长路径
    for (int i = 0; i < V; i++)
        (dist[i] == NINF)? cout << "INF ": cout << dist[i] << " ";
}

// Driver program to test above functions
int main()
{
    // Create a graph given in the above diagram.  Here vertex numbers are
    // 0, 1, 2, 3, 4, 5 with following mappings:
    // 0=r, 1=s, 2=t, 3=x, 4=y, 5=z
    Graph g(6);
    g.addEdge(0, 1, 5);
    g.addEdge(0, 2, 3);
    g.addEdge(1, 3, 6);
    g.addEdge(1, 2, 2);
    g.addEdge(2, 4, 4);
    g.addEdge(2, 5, 2);
    g.addEdge(2, 3, 7);
    g.addEdge(3, 5, 1);
    g.addEdge(3, 4, -1);
    g.addEdge(4, 5, -2);
```

```
    int s = 1;
    cout << "Following are longest distances from source vertex " << s <<" \n";
    g.longestPath(s);

    return 0;
}
```

输出结果:

1. 从源点1到其它顶点的最长距离
2. INF 0 2 9 8 10

时间复杂度：拓扑排序的时间复杂度是O(V+E).求出拓扑顺序后，对于每个顶点，通过循环找出所有邻接点，时间复杂度为O(E)。所以内部循环运行O(V+E)次。
因此算法总的时间复杂度为O(V+E)。