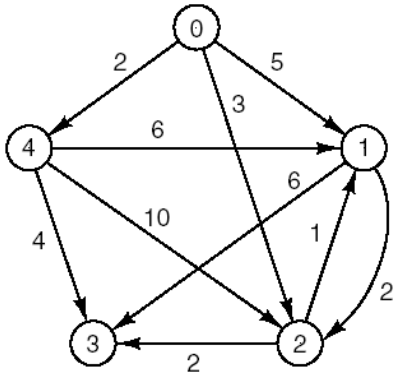
Shortest Paths
最短路径

贪心算法:最短路径

The problem of shortest paths

Given a directed graph in which each edge has a nonnegative(*非负的*) *weight*(*权*) or cost(*代价*), find a path of least total weight from a given vertex, called the *source*(*源点*), to every other vertex in the graph.



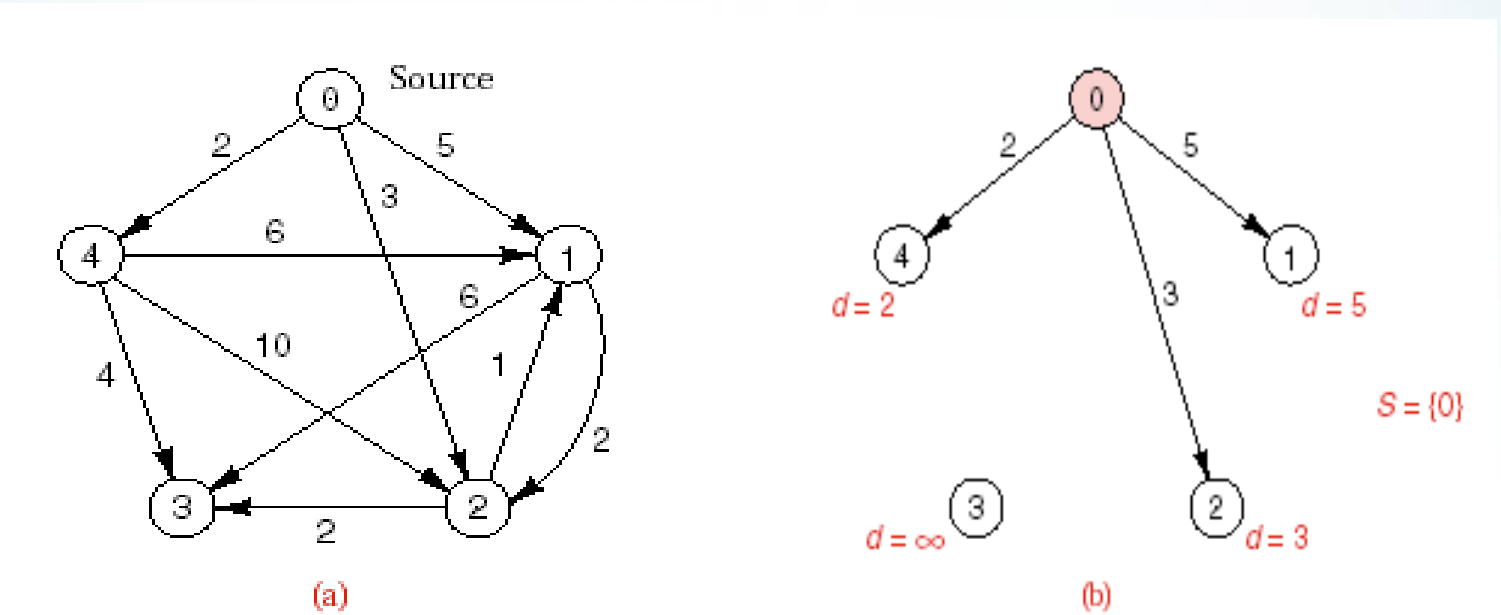
∞	5	3	∞	2
∞	∞	2	6	∞
0	∞	∞	2	∞
∞	∞	∞	∞	∞
6	6	10	4	∞

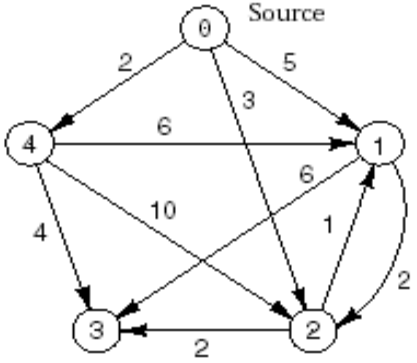
贪心算法:最短路径

- Method
  - We keep a set *S* of vertices whose closest distances to the source, vertex 0, are known and add one vertex to *S* at each stage. (采用一个辅助集合S--已找到最短路径的终点集合。S初值仅有一个顶点：v0，每次循环增加一个顶点至S集合中)。
  - We maintain a table distance that gives, for each vertex *v*, the distance from 0 to *v* along a path all of whose vertices are in *S*, except possibly the last one.(采用辅助数组Distance[0..n-1]，其中每个Distance[v]的值表示——从源点v0开始通过 S中某些顶点过达v的最短路径的长度。)
  - To determine what vertex to add to *S* at each step, we apply the *greedy* criterion(准则) of choosing the vertex *v* with the smallest distance recorded in the table distance, such that *v* is not already in S.

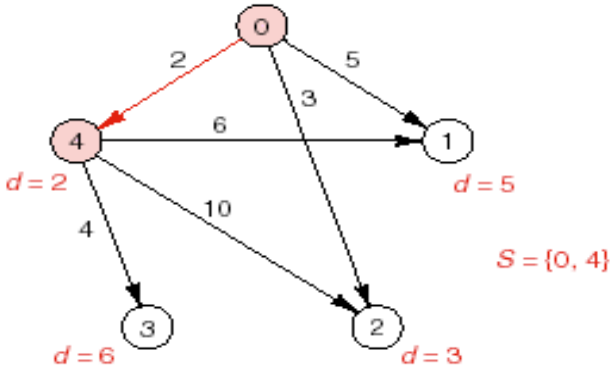
贪心算法:最短路径

- Examples

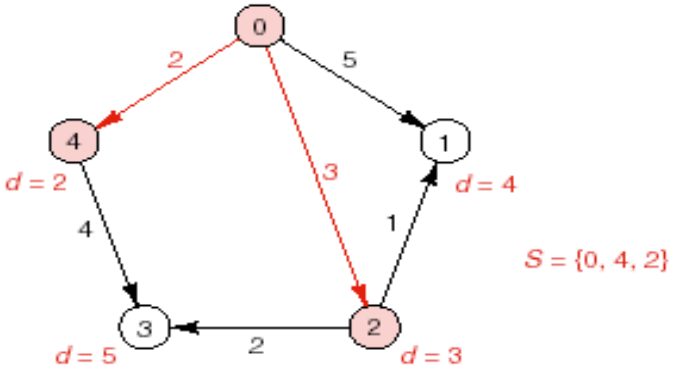



Source
0
1
2
3
4
2
5
3
6
6
10
1
4
2
2

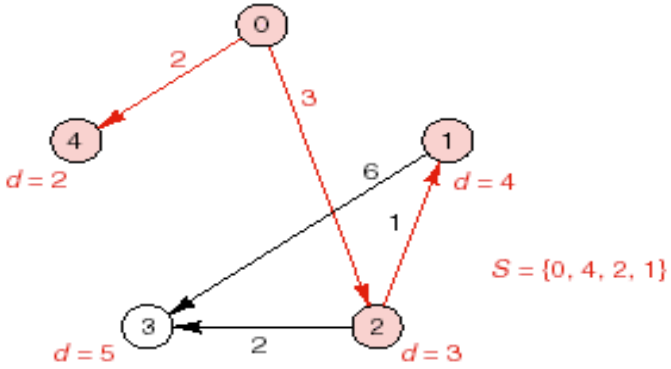
(a)


0
1
2
3
4
2
5
3
6
10
4
d = 2
d = 5
d = 6
d = 3
S = {0, 4}

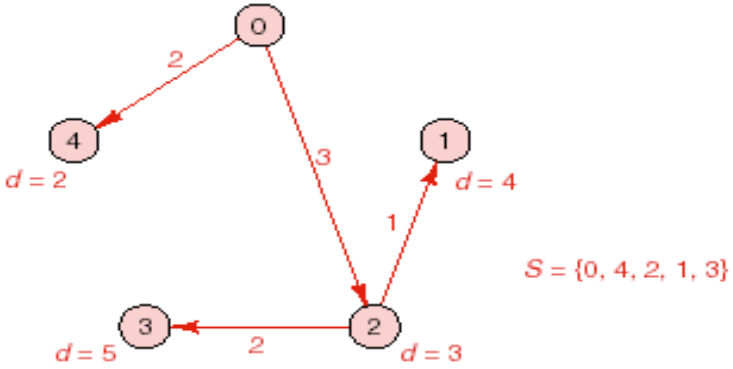
(c)


0
1
2
3
4
2
5
3
1
4
d = 2
d = 4
d = 5
d = 3
2
S = {0, 4, 2}

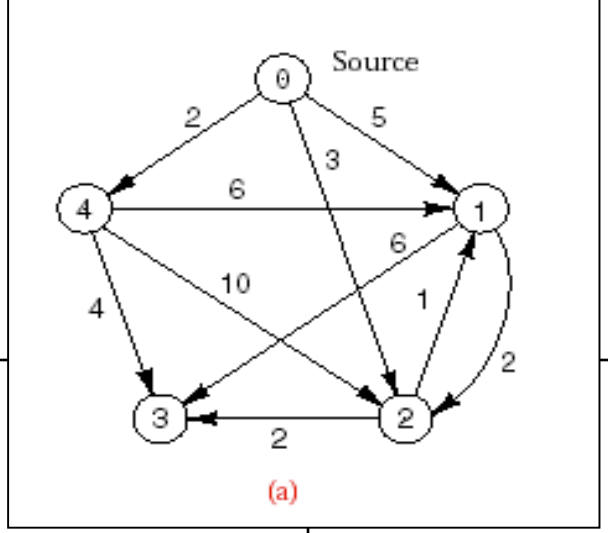
(d)


0
1
2
3
4
2
3
6
1
d = 2
d = 4
d = 5
d = 3
2
S = {0, 4, 2, 1}

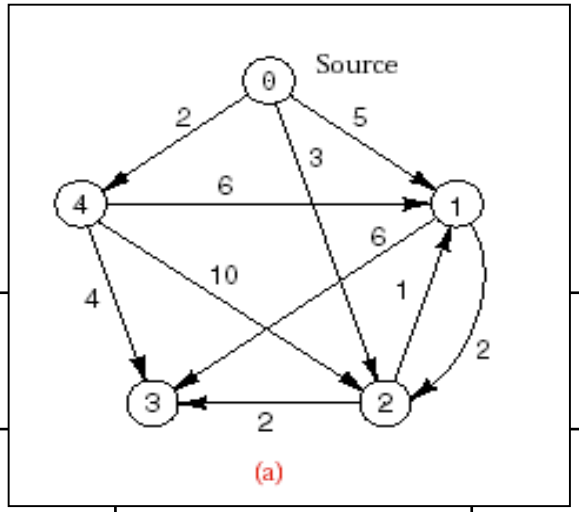
(e)


0
1
2
3
4
2
3
1
d = 2
d = 4
d = 5
d = 3
2
S = {0, 4, 2, 1, 3}

(f)

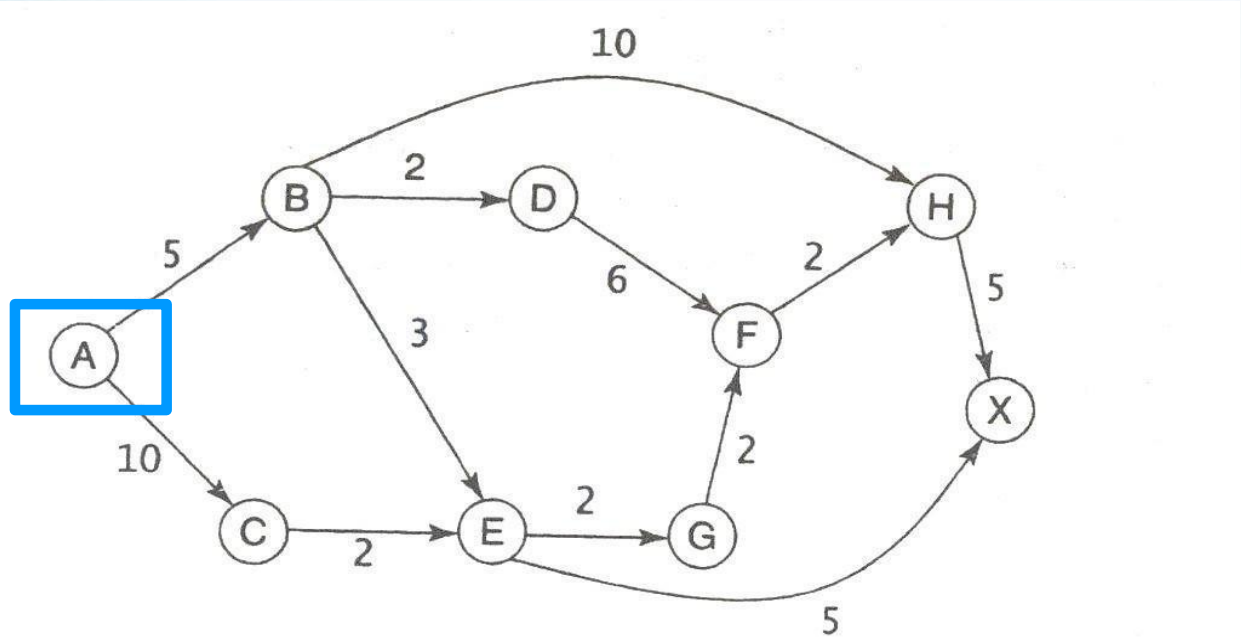


迭代次数	选择V	最短路径长度	路径	集合s{v0}	Distance[i]			
					1	2	3	4
					5 *v0*	3 *v0*	∞ *v0*	*2* *v0*
1	**v4**							
2	**v2**							
3	**v1**							
4	**v3**							



迭代次数	选择V	最短路径长度	路径	集合s{v0}	Distance[i]			
					2	3	4	
				集合s{v0}	5 v0	3 v0	∞ v0	2 v0
1	v4	2	V0->v4	{v0,v4}	5 v0	3 v0	6 v4	
2	v2	3	V0->v2	{v0,v4,v2}	4 v2		5 v2	
3	v1	4	V0->v2->v1	{v0,v4,v2,v1}			5 v2	
4	v3	5	V0->v2->v3	{v0,v4,v2,v1,v5}				

贪心算法:最短路径



贪心算法:最短路径

```
template <class Weight, int graph_size>
class Digraph {
public:
        // Add a constructor and methods for Digraph input and output.
void set_distances(Vertex source, Weight distance[]) const;
protected:
        int count;
        Weight adjacency[graph_size][graph_size];
};
```

贪心算法:最短路径

```
template <class Weight, int graph size>
void Digraph<Weight, graph_size> :: set_distances(Vertex source,
        Weight distance[]) const
/* Post: The array distance gives the minimal(最小的) path weight from
        vertex source to each vertex of the Digraph . */
{
        Vertex v, w; bool found[graph_size]; // Vertices found in S
        for (v = 0; v < count; v++) {
          found[v] = false;
          distance[v] = adjacency[source][v];
        }
```

贪心算法:最短路径

```
found[source] = true;
// Initialize(初始化) with vertex source alone in the  set S.
distance[source] = 0;
for (int i = 0; i < count; i++) { // Add one vertex v to S on each pass(趟).
        Weight min = infinity(无穷大);
        for (w = 0; w < count; w++)
        if (!found[w])
           if (distance[w] < min) {       v = w;     min = distance[w];          }
        found[v] = true;//找出具有最短distance值的v
        for (w = 0; w < count; w++)
           if (!found[w])
                   if (min +adjacency[v][w] < distance[w])
                           distance[w] = min + adjacency[v][w];
}   }
```

贪心算法:最短路径

- 单个源的最短路径
- 单个目标的最短路径
  - Revert every edge ➔ single source shortest paths
- 给定一对顶点间的Shortest path
- 图中任意顶点对间的最短路径
  - Using every vertex as the source, and calling the single source shortest paths solution to achieve the results
  - Floyd algorithm (Pls referring something else)