《算法设计与分析》

第七章 分枝一限界法

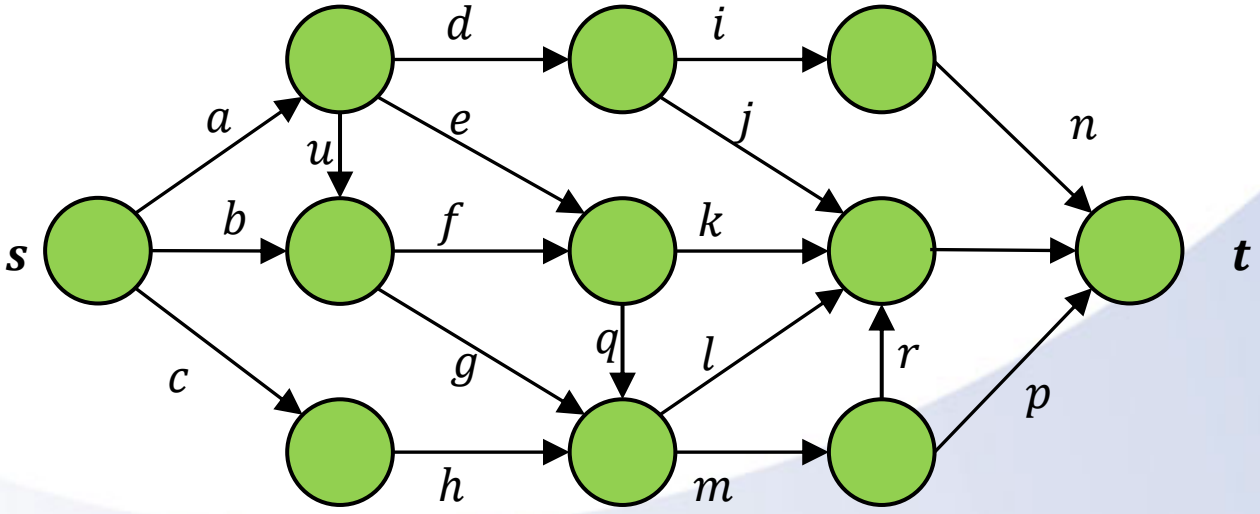
马丙鹏

2023年11月20日

中国科学院大学
University of Chinese Academy of Sciences

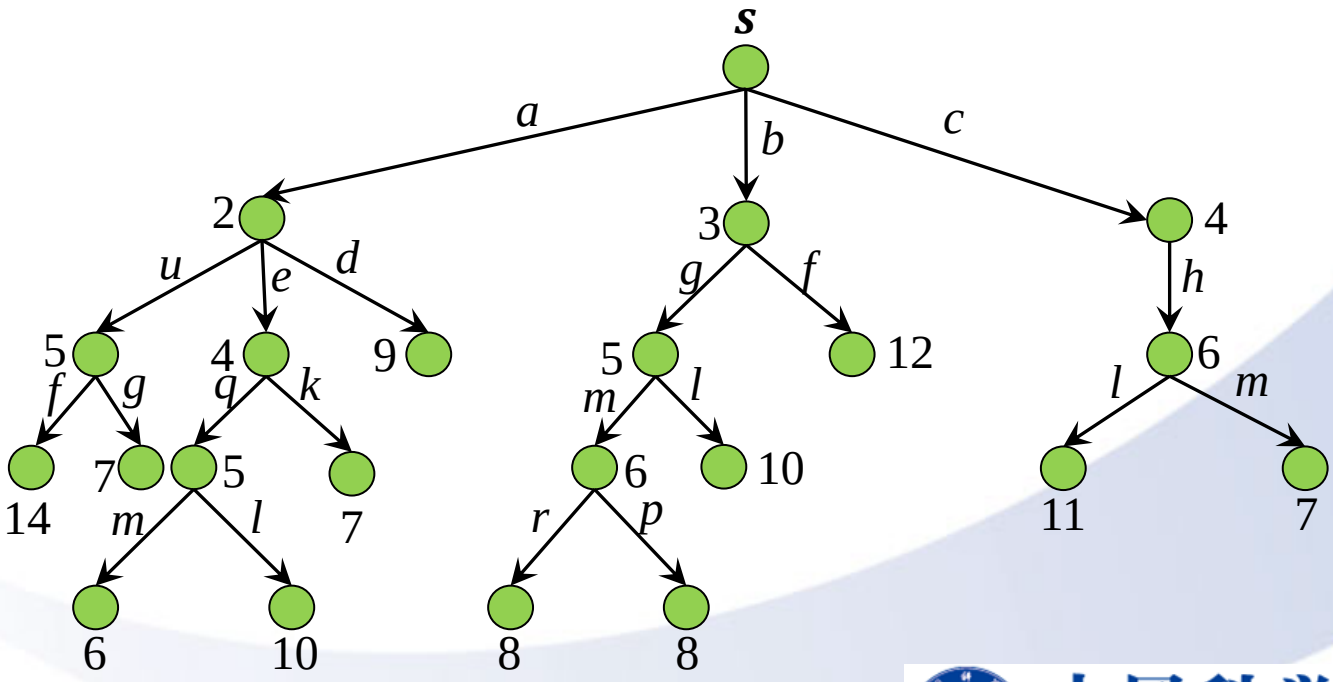
7.7 单源最短路径问题

- 问题描述
  - 在下图所给的有向图G中，每一边都有一个非负边权。
  - 求图G的从源顶点s到目标顶点t之间的最短路径



7.7 单源最短路径问题

- 分枝限界法求解
  - 用优先队列式分枝限界法解有向图G的单源最短路径问题产生的解空间树
  - 每一个结点旁边的数字表示该结点所对应的当前路长



7.7 单源最短路径问题

■ 算法思想

- 解单源最短路径问题的优先队列式分枝限界法用一极小堆来存储活结点表。其优先级是结点所对应的当前路长。
  - 开始：源顶点s和空优先队列
  - 遍历：
    - ✓结点s被扩展后，它的儿子结点被依次插入堆中。
    - ✓此后，算法从堆中取出具有最小当前路长的结点作为当前扩展结点，并依次检查与当前扩展结点相邻的所有顶点。

7.7 单源最短路径问题

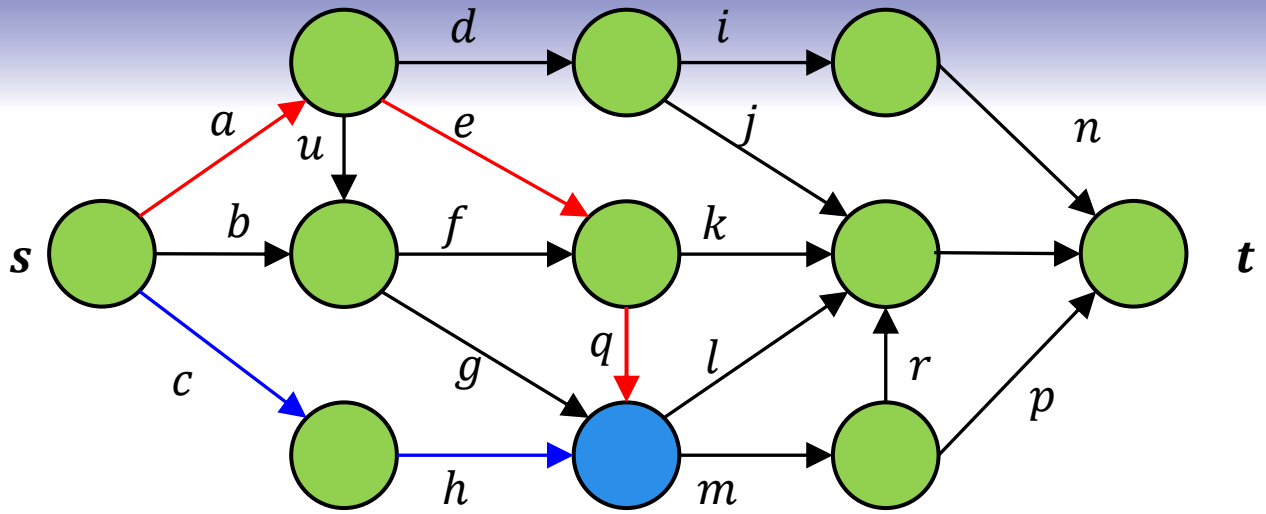
■ 算法思想

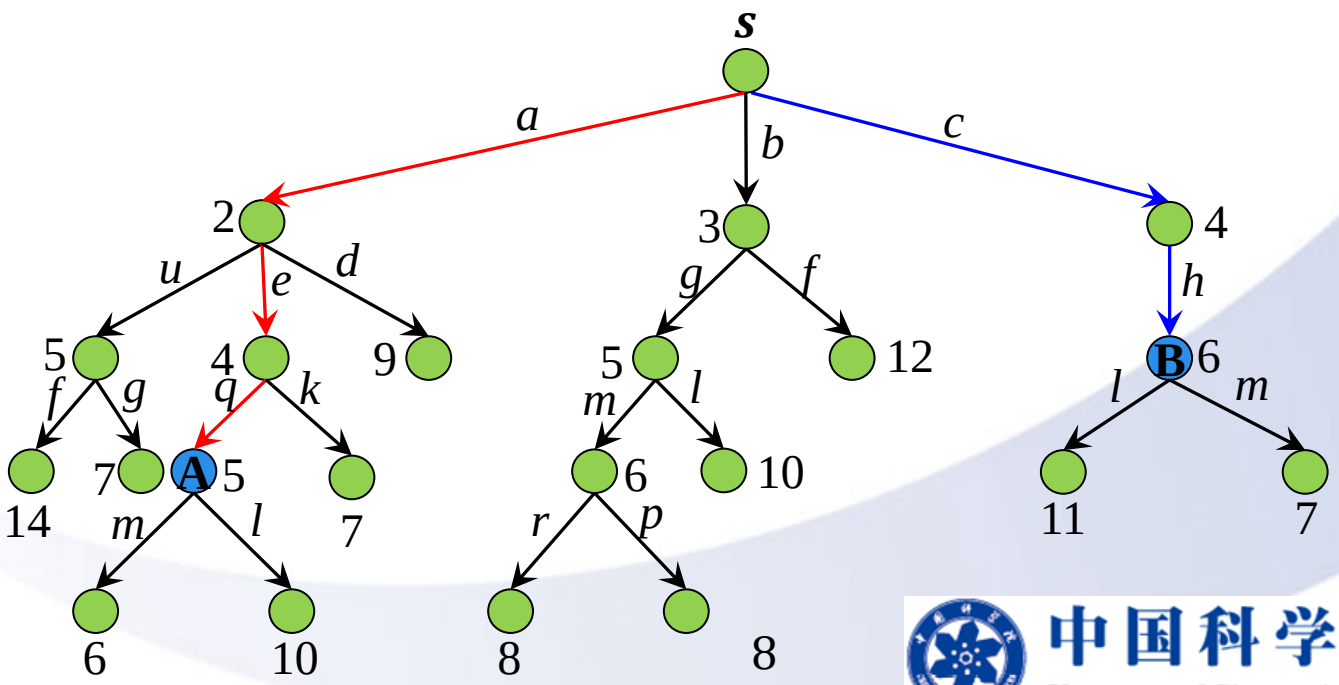
- 解单源最短路径问题的优先队列式分枝限界法用一极小堆来存储活结点表。其优先级是结点所对应的当前路长。
  - 遍历：
    - ✓如果从当前扩展结点i到顶点j有边可达，且从源出发，途经顶点i再到顶点j的所相应的路径的长度小于当前最优路径长度，则将该顶点作为活结点插入到活结点优先队列中。
  - 结束：扩展过程一直继续到活结点优先队列为空时为止。

7.7 单源最短路径问题

■ 剪枝策略

- 在扩展结点的过程中，一旦发现一个结点的下界不小于当前找到的最短路长，则剪去以该结点为根的子树。
- 在算法中，利用结点间的控制关系进行剪枝。从源顶点s出发，2条不同路径到达图G的同一顶点。由于两条路径的路长不同，因此可以将路长长的路径所对应的树中的结点为根的子树剪去。
- 例如，例中从s出发经边a、e、q（路长5）和经c、h（路长6）的两条路径到达G的同一顶点。
- 但在解空间树中，这两条路径相应于解空间树的2个不同的结点A和B。
- 由于A的路长较小，故可将以结点B为根的子树剪去。此时称结点A控制了结点B。


s
t
a
b
c
d
e
f
g
h
i
j
k
l
m
n
p
q
r
u


s
a
b
c
2
3
4
u
e
d
g
f
h
5
4
9
5
12
B
6
f
g
q
k
m
l
l
m
14
7
A
5
7
6
10
11
7
m
l
r
p
6
10
8
8

7.7 单源最短路径问题

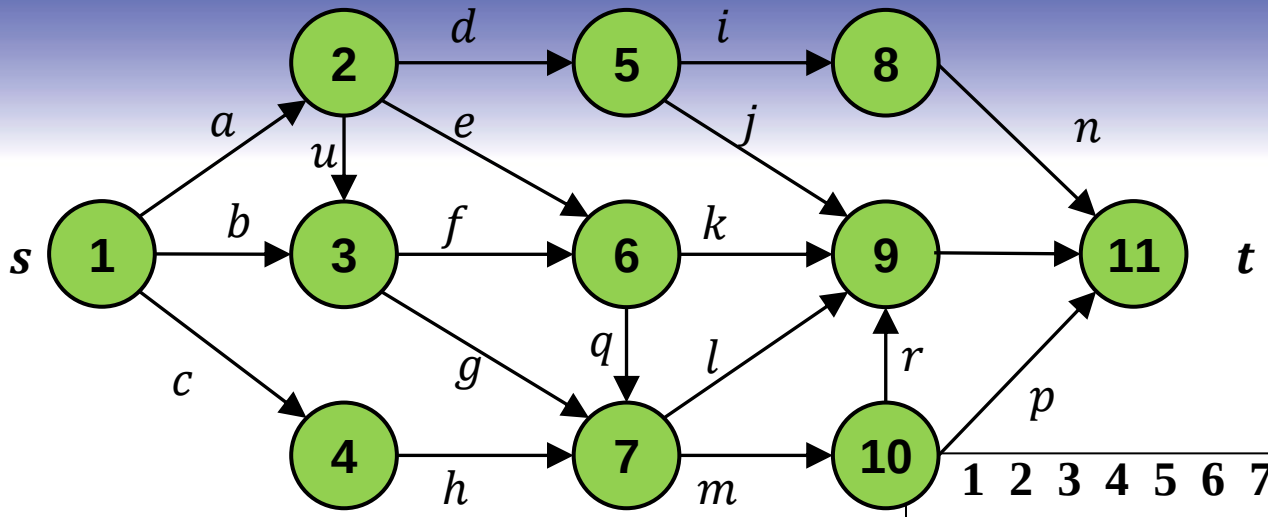
■ 算法设计

```
static float[][]a  //图G的邻接矩阵
static float []dist //源到各顶点的距离
static int []p //源到各顶点的路径上的前驱顶点
HeapNode //最小堆元素
{  int i; //顶点编号
   float length;//当前路长
  ……
 }
```

```
while (true) {//搜索问题的解空间
    for (int j = 1; j <= n; j++)
      if ((a[enode.i][j]<Float.MAX_VALUE)&&
        (enode.length+a[enode.i][j]<dist[j])) {
        // 顶点i到顶点j可达，且满足控制约束
        dist[j]= enode.length+c[enode.i][j];
        p [j]= enode.i;
        // 加入活结点优先队列
        HeapNode node=new HeapNode(j,dist[j]);
        heap.put(node);
       }
    // 取下一扩展结点
    if ( heap.isEmpty( ) ) break;
    else enode=(HeapNode)heap.removeMin();
    }
}
```

顶点i和j间有边，且此路径长小于原先从原点到j的路径长

实例：



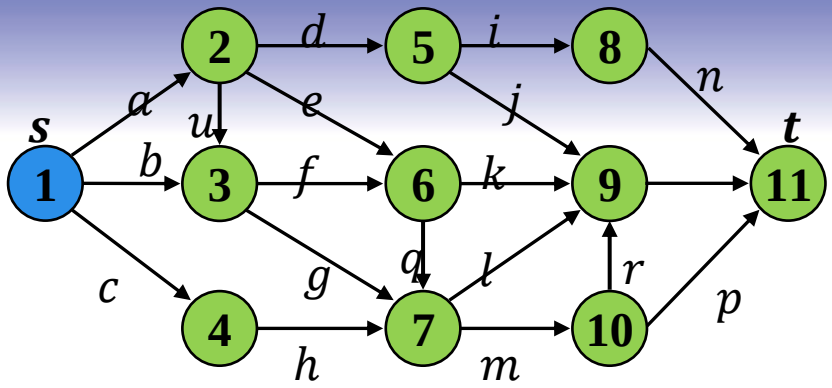
优先队列：

	1	2	3	4	5	6	7	8	9	10	11
dist	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

	1	2	3	4	5	6	7	8	9	10	11
1	0	2	3	4	∞	∞	∞	∞	∞	∞	∞
2	∞	0	3	∞	7	2	∞	∞	∞	∞	∞
3	∞	∞	0	∞	∞	9	2	∞	∞	∞	∞
4	∞	∞	∞	0	∞	∞	2	∞	∞	∞	∞
5	∞	∞	∞	∞	0	∞	∞	50	50	∞	∞
6	∞	∞	∞	∞	∞	0	1	∞	3	∞	∞
7	∞	∞	∞	∞	∞	∞	0	∞	5	1	∞
8	∞	∞	∞	∞	∞	∞	∞	0	∞	∞	50
9	∞	∞	∞	∞	∞	∞	∞	∞	0	∞	50
10	∞	∞	∞	∞	∞	∞	∞	∞	1	0	2
11	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0

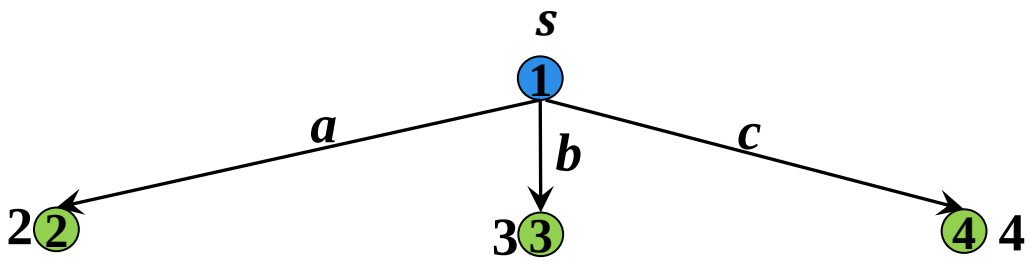
结点1到其它结点：

	1	2	3	4	5	6	7	8	9	10	11
dist	0	2	3	4	∞	∞	∞	∞	∞	∞	∞
p		1	1	1							



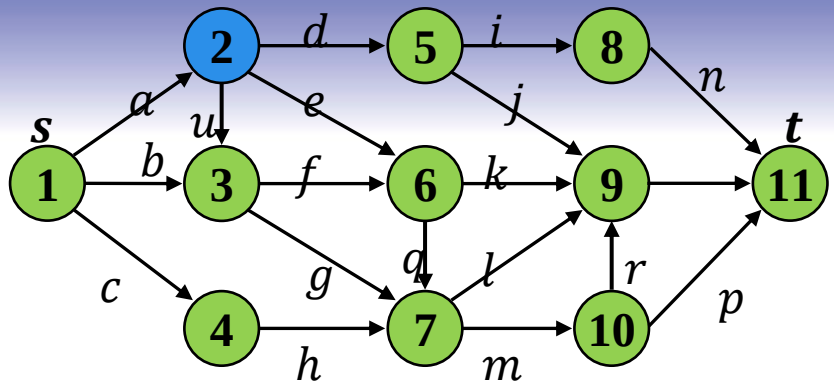
优先队列：

2	3	4		



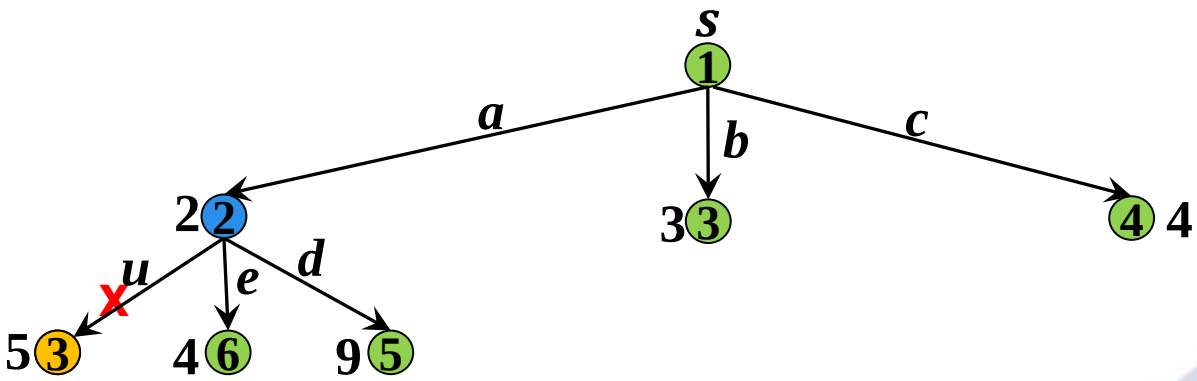
经过结点2到其它结点：

	1	2	3	4	5	6	7	8	9	10	11
dist	0	2	3	4	9	4	∞	∞	∞	∞	∞
p		1	1	1	2	2					



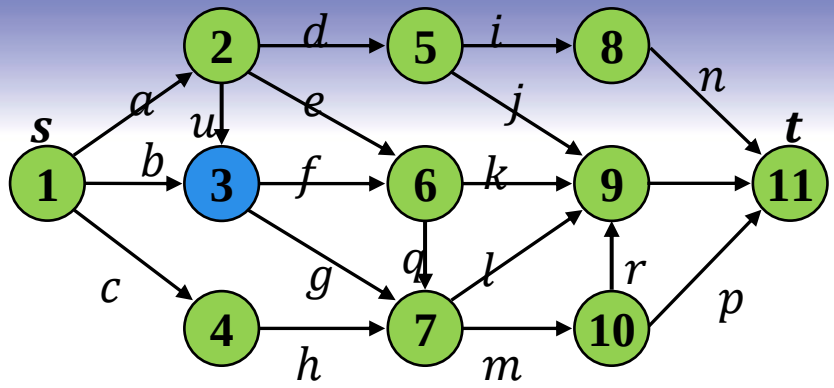
优先队列：

3	4	6	5	



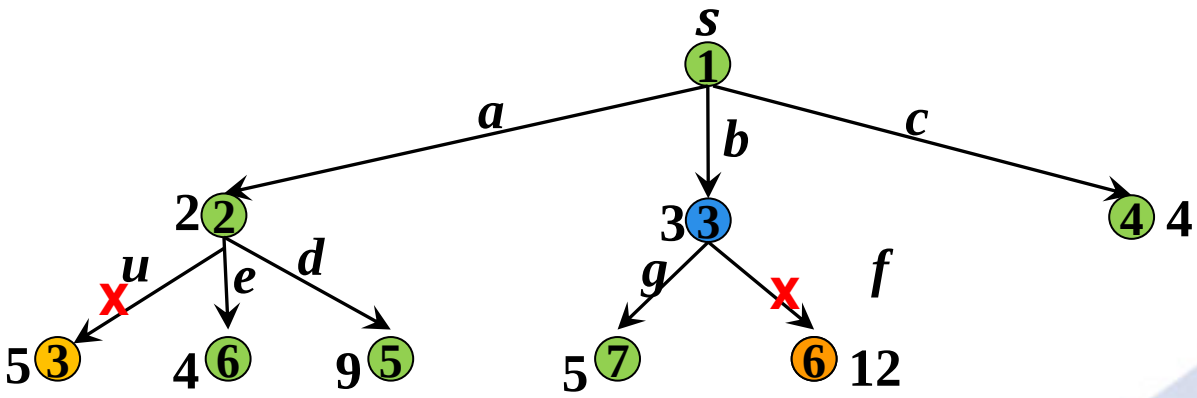
经过结点3到其它结点：

	1	2	3	4	5	6	7	8	9	10	11
dist	0	2	3	4	9	4	5	∞	∞	∞	∞
p		1	1	1	2	2	3				



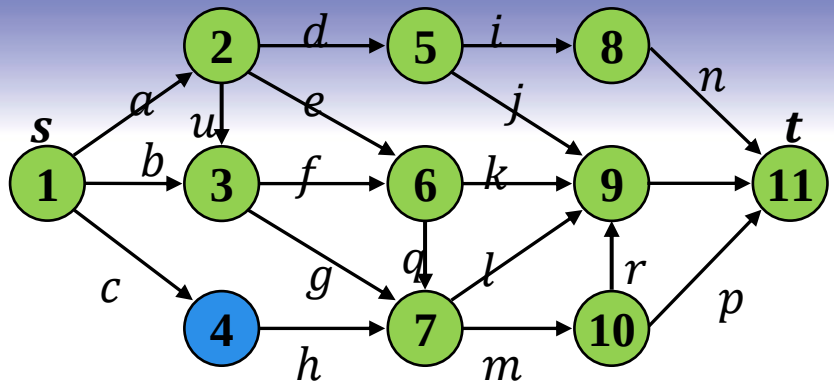
优先队列：

4	6	7	5	



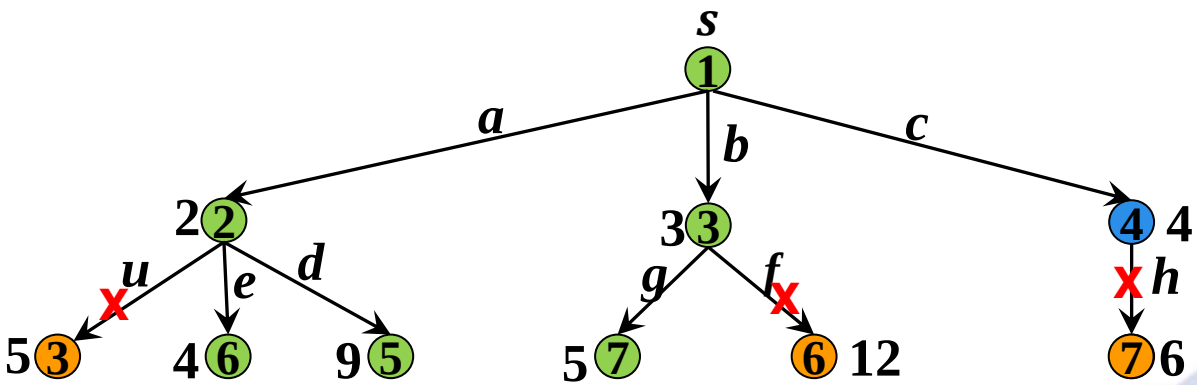
经过结点4到其它结点：

	1	2	3	4	5	6	7	8	9	10	11
dist	0	2	3	4	9	4	5	∞	∞	∞	∞
p		1	1	1	2	2	3				



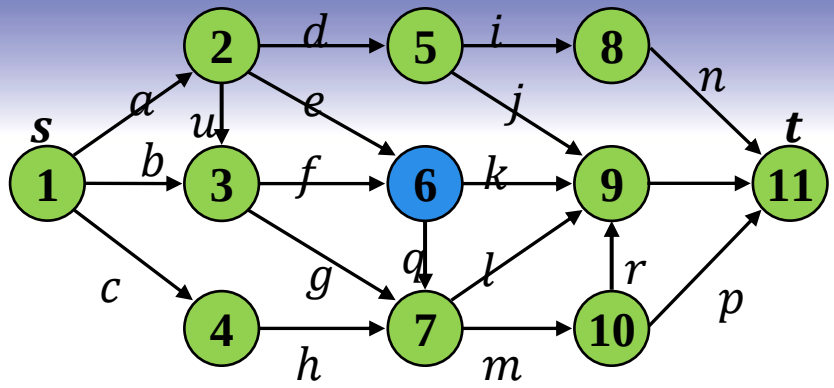
优先队列：

6	7	5		



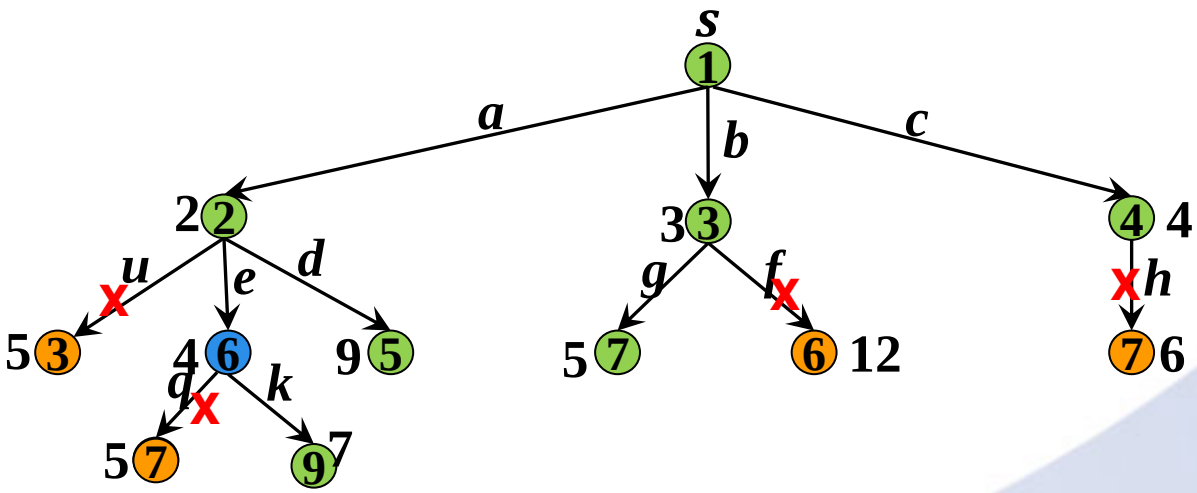
经过结点6到其它结点：

	1	2	3	4	5	6	7	8	9	10	11
dist	0	2	3	4	9	4	5	∞	7	∞	∞
p		1	1	1	2	2	3		6		



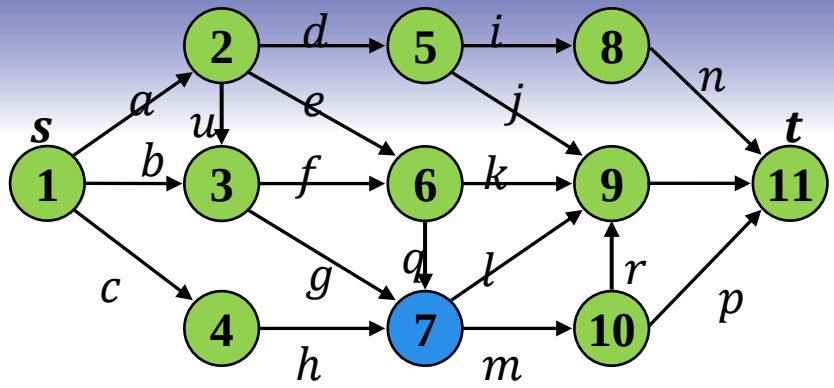
优先队列：

7	9	5		



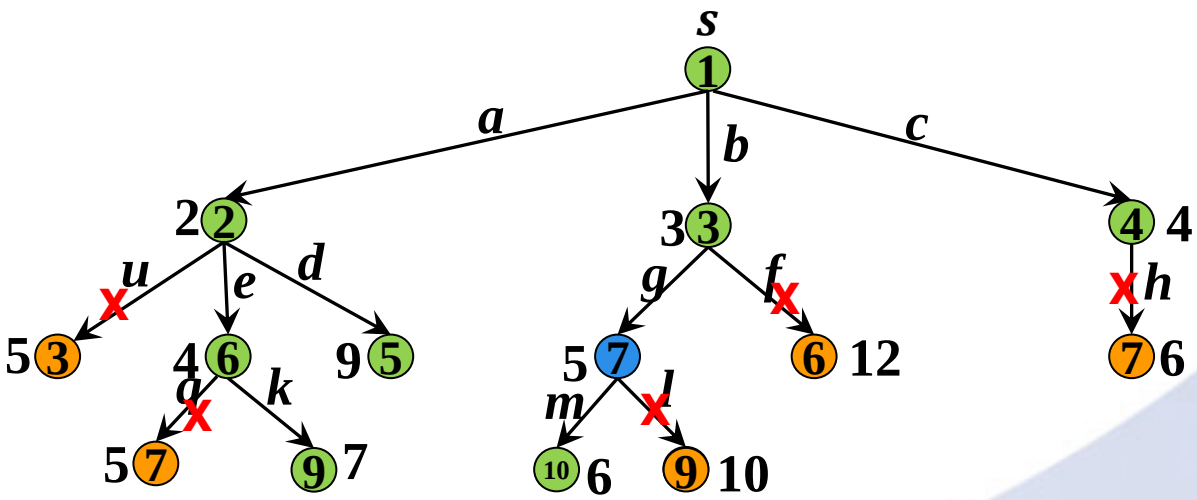
经过结点7到其它结点：

	1	2	3	4	5	6	7	8	9	10	11
dist	0	2	3	4	9	4	5	∞	7	6	∞
p		1	1	1	2	2	3		6	7	



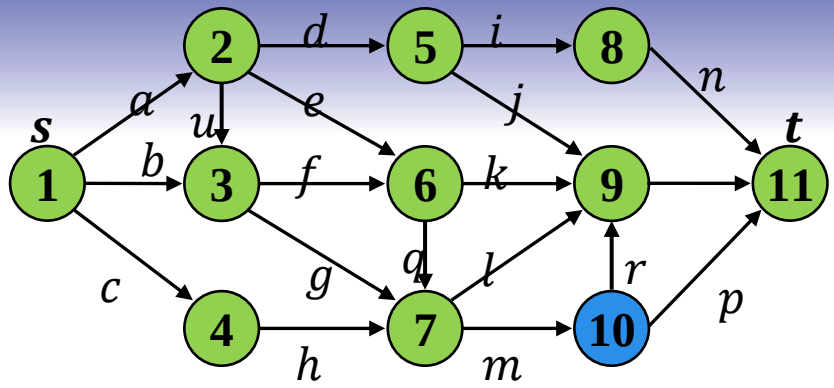
优先队列：

10	9	5		



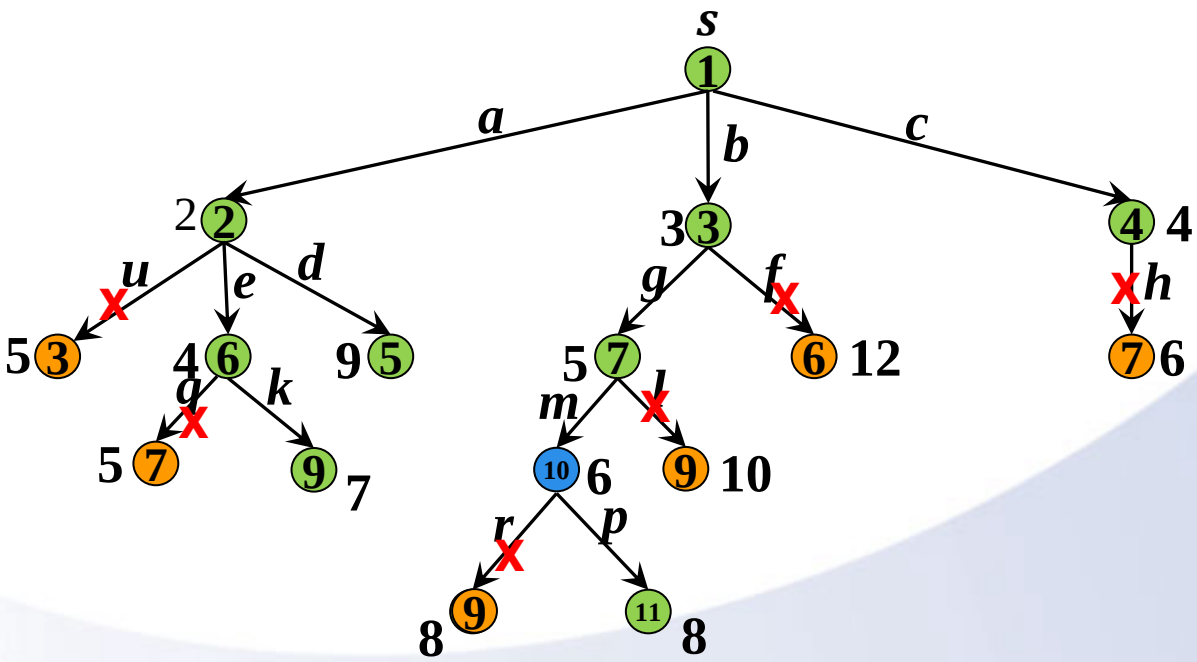
经过结点10到其它结点：

	1	2	3	4	5	6	7	8	9	10	11
dist	0	2	3	4	9	4	5	∞	7	6	8
p		1	1	1	2	2	3		6	7	10



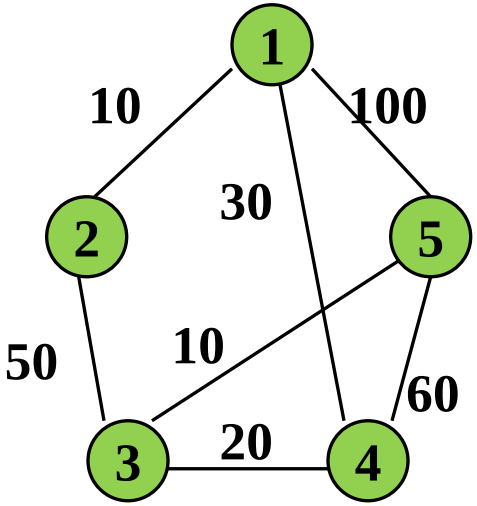
优先队列：

9	5			



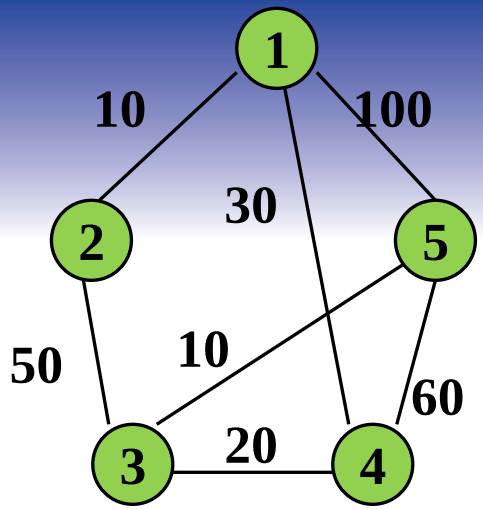
7.7 单源最短路径问题

- 求最短路径举例
  - 求源点1到汇点5的最短距离



经过结点1到其它结点：

	1	2	3	4	5
dist	0	10	∞	30	100
p		1		1	1



优先队列：

2	4	5		

经过结点2到其它结点：

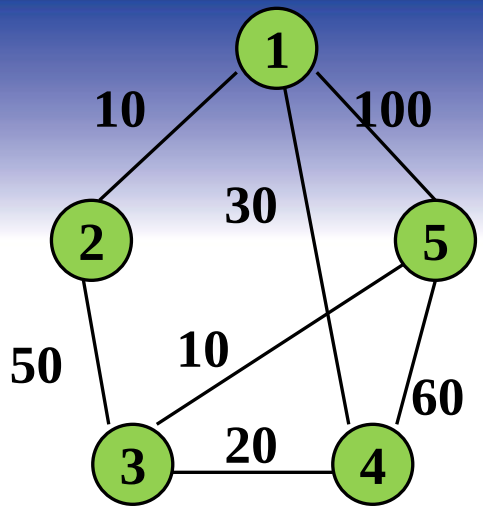
	1	2	3	4	5
dist	0	10	60	30	100
p		1	2	1	1

优先队列：

4	3	5		

经过结点4到其它结点：

	1	2	3	4	5
dist	0	10	50	30	90
p		1	4	1	4



优先队列：

3	5			

经过结点3到其它结点：

	1	2	3	4	5
dist	0	10	50	30	60
p		1	4	1	3

优先队列：

5				

7.7 单源最短路径问题

- 复杂度分析
  - 计算时间为 O(n!)
  - 需要空间为 $O(n^2)$

分枝限界法的时间性能和空间性能

- 时间性能
  - 分枝限界法和回溯法实际上都属于蛮力穷举法，当然不能指望它有很好的最坏时间复杂性，遍历具有指数阶个结点的解空间树，在最坏情况下，时间复杂性肯定为指数阶。
  - 与回溯法不同的是，分枝限界法首先扩展解空间树中的上层结点，并采用限界函数，有利于实行大范围剪枝，同时，根据限界函数不断调整搜索方向，选择最有可能取得最优解的子树优先进行搜索。
  - 所以，如果选择了结点的合理扩展顺序以及设计了一个好的限界函数，分枝界限法可以快速得到问题的解。

分枝限界法的时间性能和空间性能

- 时间性能
  - 分枝限界法的较高效率是以付出一定代价为基础的，其工作方式也造成了算法设计的复杂性。
    - 首先，一个更好的限界函数通常需要花费更多的时间计算相应的目标函数值，而且对于具体的问题实例，通常需要进行大量实验，才能确定一个好的限界函数；
    - 其次，由于分枝限界法对解空间树中结点的处理是跳跃式的，因此，在搜索到某个叶子结点得到最优值时，为了从该叶子结点求出对应的最优解中的各个分量，需要对每个扩展结点保存该结点到根结点的路径，或者在搜索过程中构建搜索经过的树结构，这使得算法的设计较为复杂；

分枝限界法的时间性能和空间性能

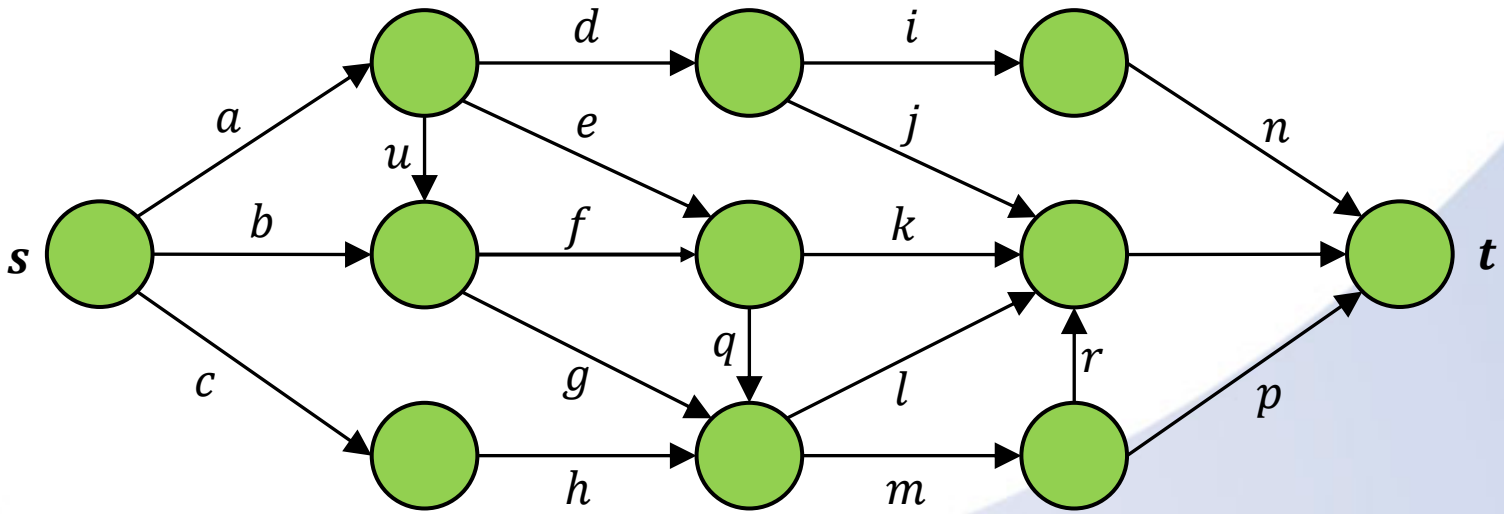
- 时间性能
  - 分枝限界法的较高效率是以付出一定代价为基础的，其工作方式也造成了算法设计的复杂性。
    - 再次，算法要维护一个待处理结点表PT，并且需要在表PT中快速查找取得极值的结点，等等。这都需要较大的存储空间，在最坏情况下，分枝限界法需要的空间复杂性是指数阶。

分枝限界法的时间性能和空间性能

- 空间性能
  1. 回溯法占用内存：O(解空间的最大路径长度)
     - 对子集问题，需要O(n)的内存空间
     - 对排列问题，需要O(n)的内存空间
  2. 分枝限界法占用内存：O(解空间大小)
     - 对子集问题，需要O(2n)的内存空间
     - 对排列问题，需要O(n!)的内存空间

作业-课后练习26

- 单源最短路径问题
  - 在下图所给的有向图G中，每一边都有一个非负边权。
  - 求图G的从源顶点s到目标顶点t之间的最短路径



作业-课后练习26

- 单源最短路径问题
  - 第一组测试参数

```
const int n = 6;    //图顶点个数加1
int c[n][n] = {{0,0,0,0,0,0}, {0,0,2,3,5000,5000},
{0,5000,0,1,2,5000}, {0,5000,5000,0,9,2},
{0,5000,5000,5000,0,2}, {0,5000,5000,5000,5000,0}}; //
图的邻接矩阵
```

  - 第二组测试参数

```
const int n = 5;   //图顶点个数加1
int c[][n] = {{0,0,0,0,0}, {0,0,2,3,5000}, {0,5000,0,1,2},
{0,5000,5000,0,9}, {0,5000,5000,5000,0}};
```

作业-课后练习26

- 单源最短路径问题
  - 第三组测试参数

```
const int n = 12;   //图顶点个数加1
int c[n][n] = {{0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,2,3,4,inf,inf,inf,inf,inf,inf,inf},
{0,inf,0,3,inf,7,2,inf,inf,inf,inf,inf},
{0,inf,inf,0,inf,inf,9,2,inf,inf,inf,inf},
{0,inf,inf,inf,0,inf,inf,2,inf,inf,inf,inf},
{0,inf,inf,inf,inf,0,inf,inf,3,3,inf,inf},
{0,inf,inf,inf,inf,inf,0,1,inf,3,inf,inf},
{0,inf,inf,inf,inf,inf,inf,0,inf,5,1,inf},
{0,inf,inf,inf,inf,inf,inf,inf,0,inf,inf,3},
{0,inf,inf,inf,inf,inf,inf,inf,inf,0,inf,2},
{0,inf,inf,inf,inf,inf,inf,inf,inf,2,inf,2},
{0,inf,inf,inf,inf,inf,inf,inf,inf,inf,inf,0},}; //邻接矩阵
```

End