Chapter III Algorithm Design and Analysis

1 复杂度分析

一个算法通常包含三个步骤：分配步骤、算术步骤、逻辑步骤。
衡量算法性能的三个基本途径：经验分析、一般情况分析、最坏情况分析
它们各有优缺点(p57)

我对nlogn,mlogm,mlogC,mlogU的理解：如采用如下方法存储数据

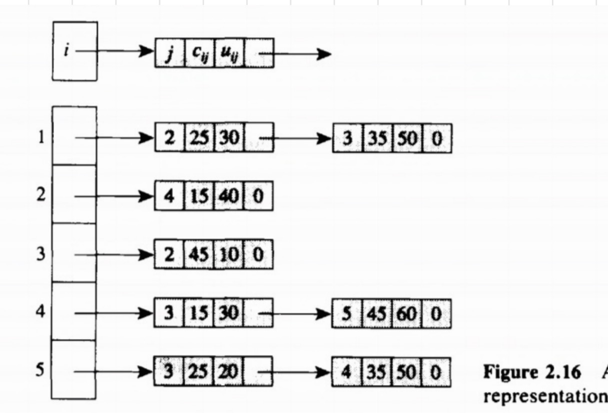


Figure 2.16 Adjacency list representation of the network example.

需 $n\log n + m\log m + m\log C + m\log U$ bits

n个节点，每个需要 $\log n$ bit

m个 ub，每个需 $\log U$ bit

大O表示法：表示程序的执行时间随数据规模增长的趋势，它代表的是最坏情况下的程序执行时间。

多项式时间算法：指的是O(f(x))中的f(x)是一个关于x的多项式，例如O(nlog(n)),O(n^3),O(m+nlogC)等。
(如果一个算法的最差情况也满足此条件的话，此算法就可被称为“好”算法；如果多项式中不包含C与U，那么就称之为“强多项式算法”，否则为“弱多项式算法”。)

指数时间算法：$f(x)$不由多项式约束，如 $O(nC)$, $O(2^n)$, $O(n!)$
当算法时间以n,m,C,U为多项式界，称之为“伪多项式算法”，它是指数时间算法的一个子类。如 $O(m + nC)$ and $O(mC)$

任何多项式时间算法都渐进优于指数时间算法。

2 多项式时间算法设计

四种获得网络流问题多项式算法的途径：几何改进法，标度法，动态规划法，二分搜索.

①几何改进法

and optimal solutions. Let H be the difference between the maximum and minimum objective function values of an optimization problem. For most network problems, H is a function of n, m, C, and U. For example, in the maximum flow problem $H = mU$, and in the minimum cost flow problem $H = mCU$. We also assume that

个人关于H=mU与H=mCU的看法：
最大流问题中的两极端：0和所有m个弧都通过了upperbound个流.
最小费用流问题两极端：0和所有m个弧都通过了花费为c的ub个流.

?
存疑

定理：若一个算法满足 $(z^k - z^{k+1}) \geq \alpha(z^k - z^*)$ （背景为minimization）

第k次迭代得到的目标函数值　　常量，(0,1)　　最小目标函数值

则这个算法定会在O((logH)/α)次迭代后结束。

可总结为：拥有几何收敛率的网络算法是多项式算法。

②标度法

能在大多数网络问题中取得最佳的最差情况运行时间.
最简单的标度法形式——位标度：

位标度法的特点：
将数据用二进制数表示，并分别用P1,P2...表示包含二进制数第一位、第二位...的情况.
P(k-1)的最优解作为P(k)的初始解.
当从一个较好初始解开始优化比从头开始优化更有效率时，标度法更佳实用.

案例分析：

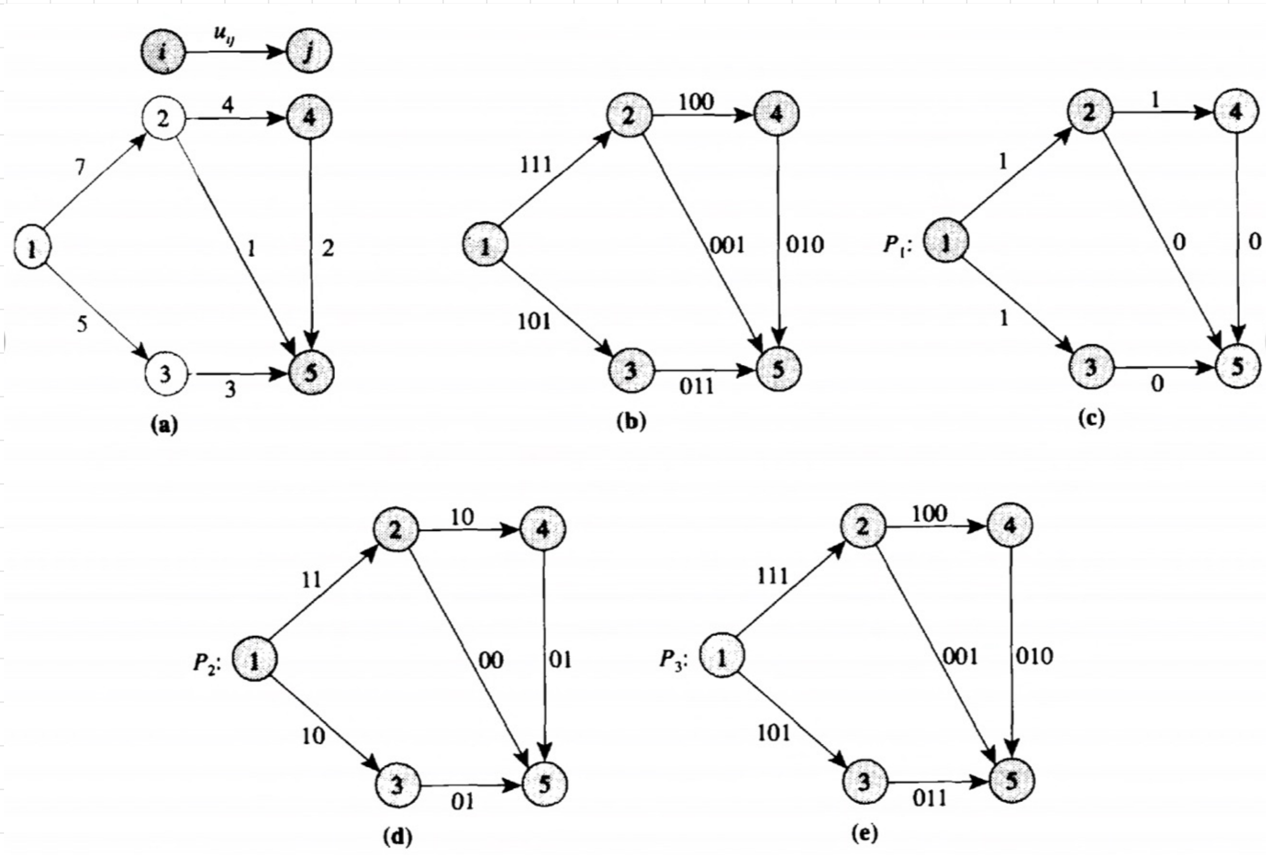


Figure 3.2 Examples of a bit-scaling technique: (a) network with arc capacities; (b) network with binary expansions of arc capacities; (c)–(e) problems P_1, P_2, and P_3.

将这个网络中的最大容量U所占字节数（log(U)=K），作为最大二进制位数，其余流的最大容量如不足则用0补齐。

对同一条弧的容量，P(k)是P(k-1)加上0或1的两倍。

```
algorithm bit-scaling;
begin
    obtain an optimal solution of P1;
    for k: = 2 to K do
    begin
        reoptimize using the optimal solution of Pk-1 to obtain an optimal solution of Pk;
    end;
end;
```

Figure 3.3 Typical bit-scaling algorithm.

标度法的变式应用广泛且稳定性强，且对最小最大流很有效，因为：p(1)通常很好解；P(k-1)的解是P(k)的一个良好初始解，从P(k-1)出发就能很容易得到P(k)的最优解；重新优化问题的复杂度是O(log(C))或O(log(U))，重优化只需比优化稍微有效率一点。

另一种标度法：P(1),P(2)⋯P(K)包含的数据都是原始数据，但求解的时候采用近似求解，误差记为Δ，delta在后续过程中几何收敛至0，通过某种方法每次修正Δk的值，使其达到最优。

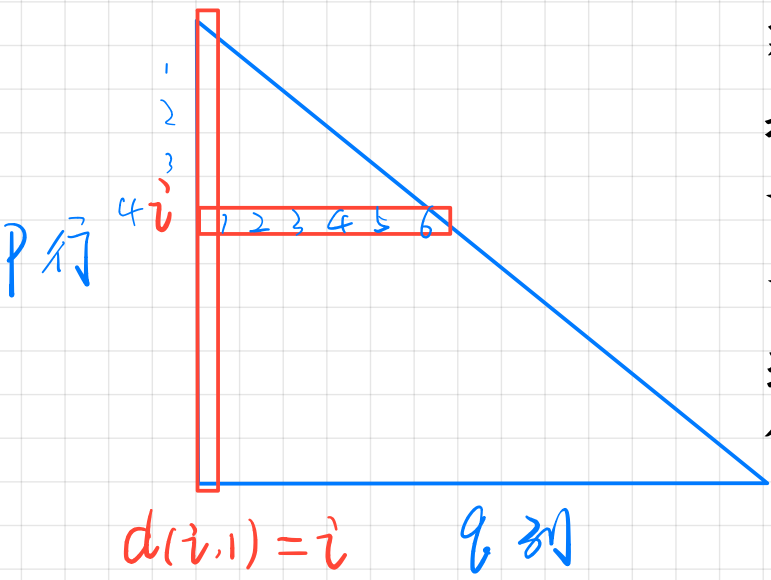
③动态规划

通过几个例子阐述这种方法：

(一)计算二项式系数

$(a+b)^n = C_n^0 a^n + C_n^1 a^{n-1} b + ... + C_n^i a^{n-i} b^i + ... + C_n^n b^n$

当$n > k > 0$时，$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$，$C_n^k = n!/(n-k)!k!$



建立下三角矩阵

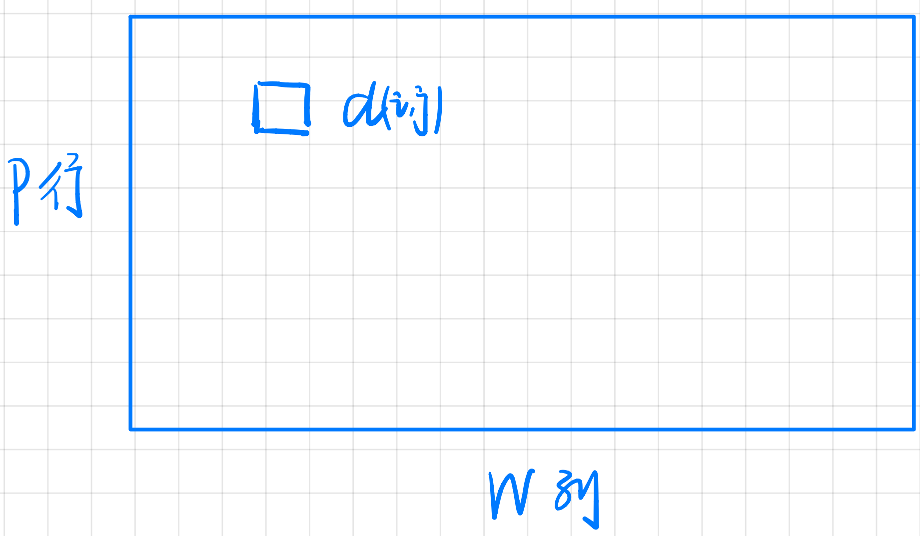
填充整个下三角矩阵表：以从1行到p行的顺序，对于每一行i，以从1到i的顺序扫描它的列。?

可以从算出第一列的值开始，因iC1=i.

这样，填完表之后，便能根据 $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$，便捷地得出每个d(i,j)的值。

(二)背包问题

建立一个p*W矩阵



则d(i,j)表示：当把可选物品限制在1~i之间，而重量限制在j时，所取物品的最大效用

建立一个递归关系，将d(i,j)的值用前面已算出的值表述出来，即：

$$d(i,j) = \max\{d(i-1,j), u_i + d(i-1, j-w_i)\}.$$

由此，可以总结动态规划法的思想：在这两个动态规划示例中，我们按照升序扫描表中的行，对于每个固定行我们按照升序扫描列。通常，只要递归关系允许我们从已经计算的条目中确定递归中需要的条目，我们就可以按升序或降序扫描表的行和列。

将动态规划问题视为由阶段与状态组成，通常阶段与时间点有关，为了在这个阶段和状态框架中重新概念化我们的表格方法，我们将把每一行看作一个阶段，而每一行中的每一列看作该阶段中可能的状态。对于我们所考虑的二项式系数和背包应用，每个阶段都对应于一组受限制的对象(项目)：在每种情况下，阶段i对应于一个只包含第i个对象的受限制问题

④二分搜索

在搜索可能值出现的区间[l,u]中，不断地取中点(l+u)/2搜索，并根据结果修改上下限，最终找出需搜索的值。

3 搜索算法

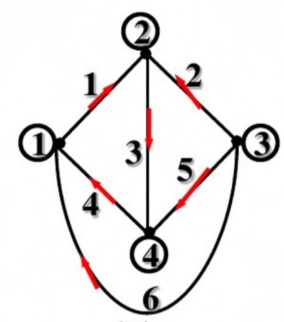
从特定点s开始，寻找出所有可达点：
将所有点归为两类——标记点和未标记点，所有被标记点就是可达点。
如果一个点i已被标记，而j未被标记，它们之间有弧(i,j)相连，则j也可以被标记，弧(i,j)被称为可接受弧，称i是j的前置点。
从原点s开始寻找所有可接受弧，直到再也找不出新的可接受弧，就找出了所有的可达点。

算法实现：

所用数据结构：节点-弧关联矩阵

```
algorithm search;
begin
    unmark all nodes in N;
    mark node s;
    pred(s) : = 0;
    next : = 1;
    order(s): = s;
    LIST : = {s}
    while LIST ≠ Ø do
    begin
        select a node i in LIST;
        if node i is incident to an admissible arc (i, j) then
        begin
            mark node j;
            pred( j) : = i;
            next : = next + 1;
            order( j) : = next;
            add node j to LIST;
        end
        else delete node i from LIST;
    end;
end;
```

Figure 3.4 Search algorithm.

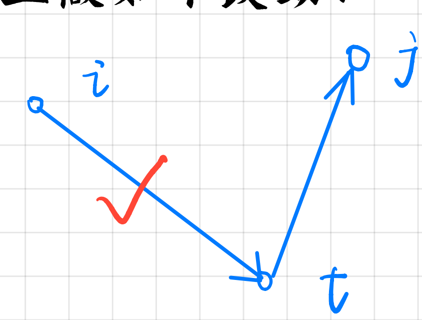


图G

$$A_a = \begin{array}{c} \\ ① \\ ② \\ ③ \\ ④ \end{array}\begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} +1 & 0 & 0 & -1 & 0 & -1 \\ -1 & -1 & +1 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 & +1 & +1 \\ 0 & 0 & -1 & +1 & -1 & 0 \end{bmatrix} \end{array}$$

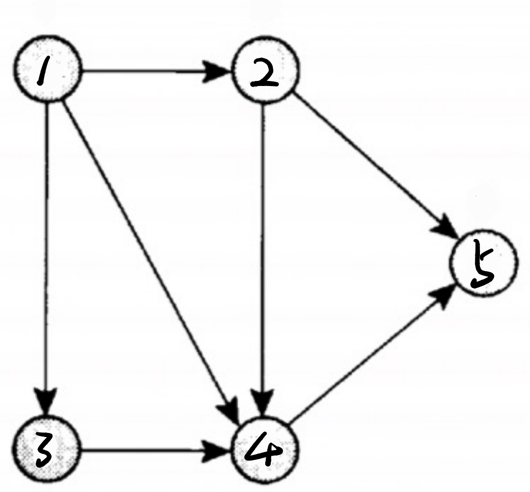
$A_{jk}=$
- +1表示支路k与节点j关联，且其方向离开节点。
- -1表示支路k与节点j关联，且其方向指向节点。
- 0表示支路k与节点j无关联。

移步GitHub查看程序实现
广度优先算法：List先进先出。此算法中，从原点s到任何其他点的路径都是最短路。
深度优先算法：List先进后出。
反向搜索(寻找点t是否与原点连接)：在搜索算法的基础上做如下改动：
1.将起始点改成t.
2.由寻找弧指向的点变成寻找哪条弧指向这个点.
3.如果i没有被标记，而j被标记了，则称(i,j)是可接受弧.



拓扑排序：如果在网络中，弧(i,j)对应的i>j，那么就是拓扑排序.

寻找拓扑排序的算法：删除没有进入弧的节点及与其连接的弧。




```
algorithm topological ordering;
begin
    for all i ∈ N do indegree(i) : = 0;
    for all (i, j) ∈ A do indegree( j) : = indegree( j) + 1;
    LIST : = Ø;
    next : = 0;
    for all i ∈ N do
        if indegree(i) = 0 then LIST : = LIST ∪ {i};
    while LIST ≠ Ø do
    begin
        select a node i from LIST and delete it;
        next : = next+1;
        order(i) : = next;
        for all (i, j) ∈ A(i) do
        begin
            indegree( j) : = indegree( j) − 1;
            if indegree( j) = 0 then LIST : = LIST ∪ { j};
        end;
    end;
    if next < n then the network contains a directed cycle
    else the network is acyclic and the array order gives a topological order of nodes;
end;
```

移步Github查看程序实现

Figure 3.8 Topological ordering algorithm.

4　流分解算法

两种表示网络中的流的方式：弧流、路径和循环流

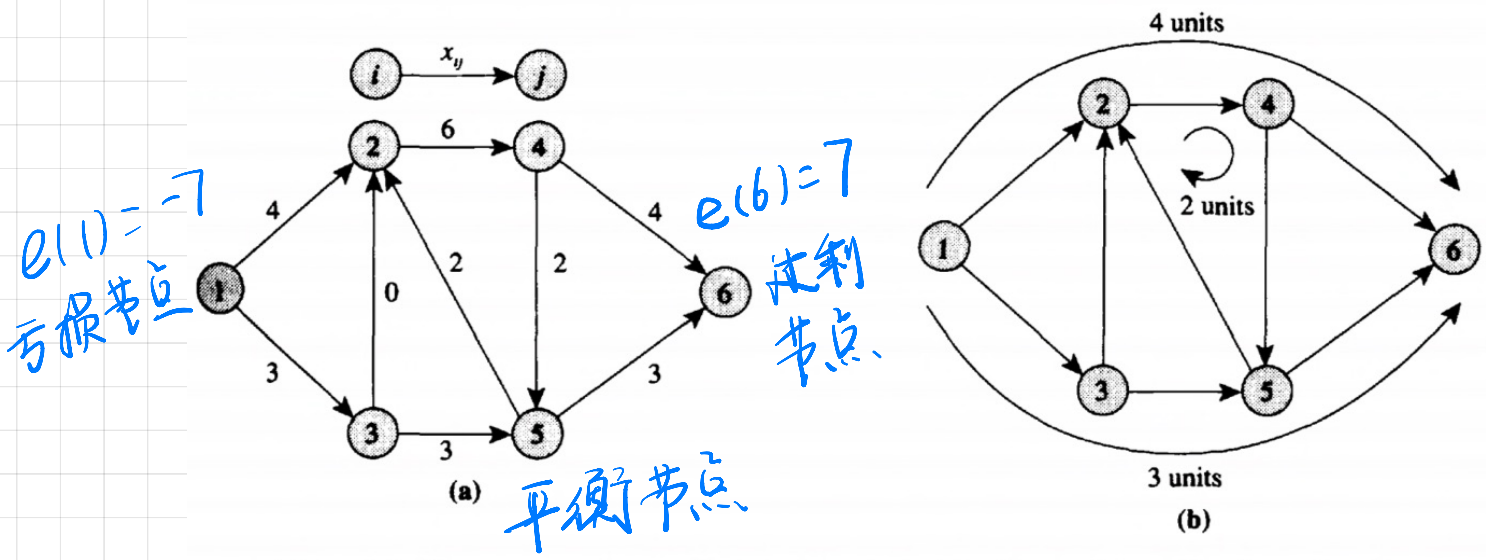


Figure 3.9 Two ways to express flows in a network: (a) using arc flows; (b) using path and cycle flows.

流分解算法：将(a)转换成(b)的算法

流分解定理：每个路径和循环流都有一个唯一的非负弧流表示。相反，每个非负弧流x可以表示为具有以下两个特性的路径和循环流.
(尽管不一定唯一)：
(a)每一条有向正流路径连接一个亏损节点和一个过剩节点.
(b) 最多$n+m$路径和循环具有非零流；其中，最多m个循环具有非零流.

演示：　亏损节点有1、5，假设从节点5开始，寻找定向路径.