

# 湖南科技大学课程教案

## (章节、专题首页)

授课教师: 王志喜

职称: 副教授

单位: 计算机科学与工程学院

课程名称	算法设计与分析
章节、专题	问题的复杂性
教学目标及基本要求	熟悉问题的复杂性、 $P$ 类问题、 $NP$ 类问题和 $NP$ 完全问题等相关概念, 了解一些典型的 $NP$ 完全问题, 掌握 $NP$ 完全性的证明方法。
教学重点	$NP$ 完全问题的相关概念, $NP$ 完全问题的证明方法
教学难点	$NP$ 完全问题的相关概念, $NP$ 完全问题的证明方法
教学内容与时间分配	(1) 可计算问题与问题的复杂性 (0.75课时) (2) 非确定性算法与 $NP$ 类问题 (0.6课时) (3) 问题变换与 $NP$ 完全类问题 (0.3课时) (4) 一些典型的 $NP$ 完全问题 (0.5课时) (5) $NP$ 完全问题的证明方法 (1.25课时) (6) 近似算法与 $P = NP$ (0.6课时) 共计4课时。
习题	第11.7.1节(基础训练题)和第11.7.2节(实验题)。

## 第11章 问题的复杂性

前述章节只讨论了算法的复杂性，本章讨论问题的复杂性，作为对复杂性理论的初步接触。

本章首先介绍问题的复杂性、 $P$ 类问题、 $NP$ 类问题和 $NP$ 完全问题等相关概念，列举一些典型的 $NP$ 完全问题，然后通过实例说明 $NP$ 完全性的证明方法。

- 可计算问题与问题的复杂性
- 非确定性算法与 $NP$ 类问题
- 问题变换与 $NP$ 完全类问题
- 一些典型的 $NP$ 完全问题
- $NP$ 完全问题的证明方法

## 11.1 可计算问题与问题的复杂性

### 11.1.1 关于可计算问题

#### 1. 可计算问题的含义

若某问题存在求解的算法，则称其为可计算的；若不存在算法但是存在求解的过程，则称其为半可计算的；若连求解的过程也不存在，则称其为完全不可计算的。

## 2. 可计算问题的分类

按照最好求解算法的计算时间分类，可以分成下列两类。

(1) 可以用多项式时间算法求解的问题。例如

- 以比较为基础的搜索问题—— $O(\log n)$
- 以比较为基础的排序问题—— $O(n \log n)$

这里给出的是最好求解算法的计算时间（后续部分会证明该结论）。

(2) 没有找到多项式时间求解算法的问题。例如

- 旅行商问题—— $O(n^2 2^n)$
- 最大团问题—— $O(2^{n/3})$
- 0/1背包问题—— $O(2^{n/2})$

这里给出的是目前最好的求解算法的计算时间，都不是多项式时间。

## 11.1.2 关于问题的复杂性

如果一个问题  $P$  的任何算法在最坏情况下所需的最少时间  $t(n) = \Omega(f(n))$ , 则称问题  $P$  的时间下界为  $f(n)$  或问题  $P$  的时间复杂性为  $f(n)$ 。问题的空间下界可以类似定义。

问题的时间下界和空间下界可以统称为问题的复杂性下界。下界理论就是专门研究问题的复杂性下界的, 与  $NP$  完全问题的研究有着紧密联系, 是复杂性理论的基础。

后续两小节介绍以比较为基础的搜索问题和排序问题的时间下界。

## 11.1.3 以比较为基础的搜索问题

以比较为基础的搜索问题的时间复杂性为 $\log n$ 。下面证明该结论。

### 1. 二叉比较树

首先考虑一下折半搜索算法的时间复杂性。可以用一个二叉比较树来分析折半搜索算法的时间复杂性。例如，对于9个元素的有序数组，折半搜索的二叉比较树如图11-1(1)所示。

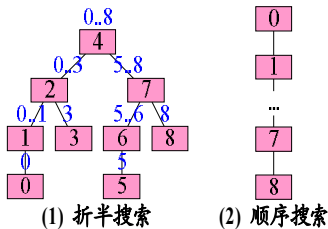
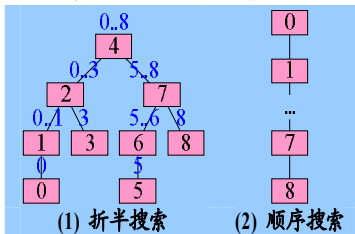


图11-1 二叉比较树

由图11-1(1)可知，一次成功的搜索至多做4次比较，而一次不成功的搜索做3次或4次比较。即当  $2^3 \leq 9 < 2^4$  时，元素比较的次数最多为4。一般地，当  $2^{k-1} \leq n < 2^k$  时，一次成功的搜索至多做  $k$  次比较，而一次不成功的搜索做  $k-1$  或  $k$  次比较，元素比较的次数最多为  $k$ 。

实际上，任何以比较为基础的搜索算法的执行过程都可以用一棵二叉比较树来描述。例如，对于9个元素的有序数组，顺序搜索的二叉比较树如图11-1(2)所示。



## 2. 搜索算法的时间下界

先回顾二叉树的一个性质，再说明搜索算法的时间下界。

【引理1】深度为 $k$ 的二叉树最多有 $2^k-1$ 个顶点。

【定理1】若 $X$ 是一个无重复元素的升序数组，则以比较为基础判断是否有 $x \in X$ 的任何算法，在最坏情况下所需的比较次数 $k$ 都满足 $k = \Omega(\log n)$ 。

【证明】通过考察模拟求解搜索问题的各种可能算法的比较树可知，在最坏情况下， $k$ 是所有顶点的最深层次，即二叉树的深度。根据二叉树的性质，深度为 $k$ 的二叉树最多有 $2^k-1$ 个顶点，从而，比较树的顶点数 $n$ 满足 $n \leq 2^k-1$ ，因此 $k \geq \log(n+1)$ ，即 $k = \Omega(\log n)$ 。【证毕】

由定理1可知，任何一种以比较为基础的搜索算法，在最坏情况下所用的时间都不可能低于 $\Theta(\log n)$ ，因此，也就不可能存在最坏情况下所需时间比折半搜索数量级更低的算法。



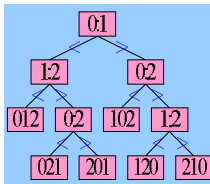
## 11.1.4 以比较为基础的排序问题

以比较为基础的排序问题的时间复杂性为 $n\log n$ 。下面证明该结论。

### 1. 二叉比较树

类似于估计搜索算法的时间下界，也可以用比较树来模拟以比较为基础的排序算法（如图11-2所示）。

假定数组中没有重复元素。在比较树的分枝顶点上，算法执行一次比较，并根据比较的结果移向某一个孩子。由于每两个元素 $x_i$ 和 $x_j$ 的比较只有 $x_i < x_j$ 或 $x_i > x_j$ 这两种可能，所以这棵树是一棵二叉树。当 $x_i < x_j$ 时进入左分枝，当 $x_i > x_j$ 进入右分枝。叶子表示算法终止。从根到叶子的每一条路径分别与一种唯一的排列相对应。因为 $n$ 个不同元素的不同排列共有 $n!$ 个，所以相应的比较树有 $n!$ 个叶子。



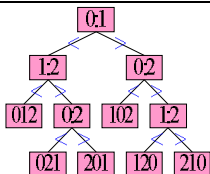


图11-2  $n=3$ 时的比较树

## 2. 排序算法的时间下界

先回顾二叉树的一个性质，再说明排序算法的时间下界。

【引理2】深度为 $k$ 的二叉树最多有 $2^{k-1}$ 个叶子（ $k \geq 1$ ）。

【定理2】若数组 $X[0..n]$ 不含重复元素，则以比较为基础的对 $X$ 排序的任何算法，在最坏情况下所需的比较次数 $k$ 满足 $k = \Omega(n \log n)$ 。

【证明】通过观察比较树可知，由根到叶子的路径描述了该叶子对应排列的生成过程，路径的长度就是经历的比较次数。因此，算法在最坏情况下所需的比较次数 $k$ 就是比较树中最长路径的长度。此时，比较树的深度为 $k+1$ ，根据二叉树的性质，深度为 $k+1$ 的二叉树最多有 $2^k$ 个叶子，从而比较树的叶子数 $n!$ 满足 $n! \leq 2^k$ ，因此

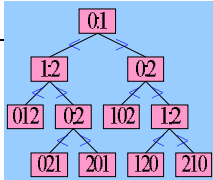
$$n! \geq n(n-1) \dots \lceil n/2 \rceil \geq \lceil n/2 \rceil^{n/2} \geq (n/2)^{(n/2)}$$

易知，当 $n \geq 4$ 时， $\log(n/2) \geq (1/2) \log n$ 。由此可得，当 $n \geq 4$ 时，

$$k \geq \log(n!) \geq (n/2) \log(n/2) \geq (1/4) n \log n$$

即 $k = \Omega(n \log n)$ 。【证毕】

由定理2可知，任何一种以比较为基础的排序算法，在最坏情况下所用的时间都不可能低于 $\Theta(n \log n)$ ，因此，也就不可能存在最坏情况下所需时间比归并排序数量级更低的算法。



## 11.1.5 关于判定问题

本章主要考虑判定问题，其他问题需要转换为判定问题以后才方便讨论。

### 1. 判定问题的含义

判定问题是答案只能是true或false的问题。例如，一个无向图是否存在不小于 $k$ 的团，一个无向图是否可 $m$ 着色，一个正整数集是否有和为 $t$ 的子集。

### 2. 转换为判定问题的方法

大多数问题都可以转换为判定问题。例如，最大团问题可以转换为判定图中是否存在不小于 $k$ 的团，最优着色问题可以转换为判定图是否可 $m$ 着色。如果知道怎样解决该判定问题，通常就可以解决原问题，常用的方法是使用二分查找。

### 3. 一个转换示例<sup>\*</sup>

以最优着色问题转换为判定问题为例说明。

(1) 判定问题。判断图 $G$ 是否可 $m$ 着色。在“回溯方法”一章中，使用回溯方法解决了该问题，这里只抄录其函数原型。

`vector<int> Chromatic(const Matrix<bool> &G, int m);`

返回值是一个整数数组，记录找到的答案，空数组表示没有满足要求的答案。

(2) 最优化问题。寻找图 $G$ 的最优着色。

```
auto Chromatic(const Matrix<bool> &G)
{
    int n = G.rows();
    vector<int> W; // 最优解
    int low = 1, high = n; // 用二分方法寻找最优解
    while(low <= high)
    {
        int m = (low + high) / 2;
        auto X = Chromatic(G, m);
        if(X.empty()) low = m + 1; // 不可m着色，需要增加颜色
        else W = X, high = m - 1; // 可m着色，尝试减少颜色
    }
    return W;
}
```

(3) 测试程序。使用图11-3所示的图。

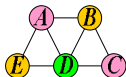


图11-3 最优着色

```
int main()
{
    Matrix<bool> G = // 图的邻接矩阵
    {
        {0, 1, 0, 1, 1},
        {1, 0, 1, 1, 0},
        {0, 1, 0, 1, 0},
        {1, 1, 1, 0, 1},
        {1, 0, 0, 1, 0}
    };
    cout << Chromatic(G) << endl; // 1 2 1 3 2
}
```

## 11.2 非确定性算法与NP问题

### 11.2.1 非确定性算法

#### 1. 成功求解

任意给定问题 $P$ 的一个候选解 $X$ ，如果 $X$ 是问题 $P$ 的满足要求的答案，则称 $X$ 问题 $P$ 的成功解。成功求解就是想办法获得问题的成功解。

## 2. 非确定性算法的含义和用途

(1) 非确定性算法的含义。前述章节主要介绍确定性算法，算法一词通常表示确定性算法。在确定性算法中，每个操作的结果是唯一定义的。非确定性算法不要求每个操作的结果是唯一定义的，即非确定性算法去除了算法定义中的唯一性限制，能力更强。例如，非确定性算法能够在常数时间内从一个集合中任意选取一个元素。

(2) 难处理的问题。一般将可由多项式时间算法成功求解的问题看作易处理的问题，而将需要超过多项式时间才能成功求解的问题看作难处理的问题。

(3) 非确定性算法的用途。有许多问题，从表面上看似乎并不比排序等问题更困难，然而至今人们还没有找到解决这些问题的多项式时间算法，也没有人能够证明这些问题是否一定需要超过多项式时间才能成功求解。非确定性算法的主要用途就是研究这类问题的计算复杂性。

- 任意给定问题 $P$ 的一个候选解 $X$ ，如果 $X$ 是问题 $P$ 的满足要求的答案，则称 $X$ 为问题 $P$ 的成功解。
- 成功求解就是想办法获得问题的成功解。



### 3. 非确定性算法的描述

(1) 三条指令。为了方便描述非确定性算法, 科学家们引入了下列三条指令, 时间耗费均为  $O(1)$ 。

- Choice( $S$ )。从集合 $S$ 中任意选取一个元素, 不关心如何选取。特别地, 用 Choice( $m, n$ )从区间 $[m, n]$ 中任意选取一个整数。
- Failure()。发出不成功信号并Stop。
- Success()。发出成功信号并Stop。

(2) 非确定性算法的2个阶段。

- 用非确定性方法选出候选解 $x$ 。
- 用确定性方法验证 $x$ 是否是该问题的成功解。

(3) 几点说明。

- 执行非确定性算法的机器称为非确定机。非确定机是一种假想的机器，现实中没有这样的机器。
- 可以认为非确定机足够聪明或足够幸运，只要问题的解存在，它总可以用最短的时间成功获得问题的解。当且仅当问题的解不存在时，非确定机求解失败。
- 也可以认为一台非确定机由若干台（足够多）独立并行工作的确定性机器组成，第一台成功求解的确定性机器发出成功信号并结束其他确定性机器的计算过程。当且仅当问题的解不存在时，每台确定性机器都发出不成功信号。
- 概率算法和非确定性算法并不等价。概率算法由确定性机器执行，非确定性算法由非确定性机器执行。当然，概率算法可以借鉴非确定算法的思想（例如使用拉斯维加斯方法和蒙特卡洛方法）。
- 使用非确定性算法，许多目前没有找到多项式时间算法的问题都可以在多项式时间内成功求解。

## 11.2.2 $P$ 类与 $NP$ 类问题

### 1. 含义

【 $P$ 类问题】用确定性算法能在多项式时间内成功求解的问题。

【 $NP$ 类问题】用非确定性算法能在多项式时间内成功求解的问题。

### 2. 性质

【 $P \subseteq NP$ 】由于一个确定性算法可以看作某个非确定性算法的特例，所以可用确定性算法在多项式时间内成功求解的问题也可用非确定性算法在多项式时间内成功求解，也就是说， $P \subseteq NP$ 。

### 11.2.3 NP问题举例

下面给出了一些问题的多项式时间内成功求解的非确定性算法, 所以这些问题都是  $NP$  的。

#### 1. 布尔表达式的可满足性问题SAT

【含义】对于由  $n$  个布尔变量  $x_1, x_2, \dots, x_n$  构成的布尔表达式  $E$ , 若存在各变量  $x_i$  ( $1 \leq i \leq n$ ) 的0/1赋值, 使得  $E=1$ , 则称布尔表达式  $E$  是可满足的。

【问题】给定一个布尔表达式, 判定它是否可满足。

【非确定性算法】

```
void SAT(function<bool(vector<bool>&)> E, vector<bool> &X)
{   int n = X.size();
    for(int i = 0; i < n; ++i)
        X[i] = Choice(0, 1);
    if(E(X) == 1) Success();
    else Failure();
}
```

【复杂性】设由 $n$ 个布尔变量构成的布尔表达式的长度为 $m$  ( $m \geq n$ )。显然, 非确定性地选取候选解耗时 $O(n)$ , 确定性地检查可满足性耗时 $O(m)$ 。所以整个算法的时间复杂性为 $O(m)$ 。

```
void SAT(function<bool(vector<bool>&)> E, vector<bool> &X)
{
    int n = X.size();
    for(int i = 0; i < n; ++i)
        X[i] = Choice(0, 1);
    if(E(X) == 1) Success();
    else Failure();
}
```

## 2. 哈密尔顿回路问题Hamilton

【问题】一个无向图是否含有哈密尔顿回路。

【非确定性算法】

```
void Hamilton(const Matrix<bool> &G, vector<int> &X)
{   int n = G.rows();
    iota(begin(X), end(X), 0); // 顶点号从0开始
    for(int i = n - 1; i > 1; --i) // 非确定性地选取一个顶点序列
        swap(X[i], X[Choice(1, i - 1)]);
    // 确定性地检查该顶点序列是否构成回路
    for(int k = 0; k < n; ++k)
    {   int i = X[k], j = X[(k + 1) % n];
        if(G(i, j) != 1) Failure();
    }
    Success();
}
```

【复杂性】显然，非确定性地选取一个顶点序列耗时 $O(n)$ ，确定性地检查该顶点序列是否构成回路耗时 $O(n)$ 。所以整个算法的时间复杂性为 $O(n)$ 。

### 3. 旅行商问题TSP

【问题】给定一个边权值非负的无向加权图 $G$ 和一个非负数 $t$ , 判定 $G$ 中是否存在费用为 $t$  (或不超过 $t$ ) 的旅行。

【非确定性算法】

```
void TSP(const Matrix<double> &G, double t, vector<int> &X)
{
    int n = G.rows();
    iota(begin(X), end(X), 0); // 顶点号从0开始
    for(int i = 1; i < n; ++i) // 非确定性地选取一个顶点序列, O(n)
        swap(X[i], X[Choice(i, n - 1)]);
    double s = 0;
    for(int k = 0; k < n; ++k) // 确定性地计算回路费用, O(n)
    {
        s += G(X[k % n], X[(k + 1) % n]);
        if(s > t) Failure();
    }
    Success();
}
```

【复杂性】显然, 非确定性地选取一个顶点序列耗时 $O(n)$ , 确定性地计算旅行费用耗时 $O(n)$ 。所以整个算法的时间复杂性为 $O(n)$ 。

## 4. 着色问题Chromatic

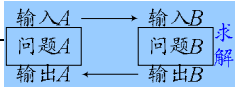
【问题】给定无向图 $G$ 和 $m$ 种颜色, 判断图 $G$ 是否可 $m$ 着色。

【非确定性算法】

```
void Chromatic(const Matrix<bool> &G, int m, vector<int> &X)
{   int n = G.rows(); // 顶点数
    for(int i = 0; i < n; ++i) // 非确定性地为顶点选取颜色,  $O(n)$ 
        X[i] = Choice(1, m);
    for(int i = 0; i < n; ++i) // 确定性地检查是否可着色,  $O(n^2)$ 
        for(int j = i + 1; j < n; ++j)
            if(G(i, j) == 1 and X[i] == X[j]) Failure();
        Success();
}
```

【复杂性】显然, 非确定性地为顶点选取颜色耗时 $O(n)$ , 确定性地检查是否可着色耗时 $O(n^2)$ 。所以整个算法的时间复杂性为 $O(n^2)$ 。





## 11.3 问题变换与NP完全问题

### 11.3.1 问题变换

将问题  $A$  变换为问题  $B$  是指下列3个步骤。

- (1) 将问题  $A$  的输入变换为问题  $B$  的适当输入。
- (2) 解出问题  $B$ 。
- (3) 把问题  $B$  的输出变换为问题  $A$  的正确解。

如果能够使用某个确定性算法在  $O(\tau(n))$  时间内完成上述第1步和第3步，则称问题  $A$  可在  $\tau(n)$  时间内变换到问题  $B$ ，简记为  $A \propto_{\tau(n)} B$ ，其中  $n$  通常是问题  $A$  的规模。若  $\tau(n)$  是一个多项式函数，则称问题  $A$  可在多项式时间内变换到问题  $B$ ，简记为  $A \propto B$ 。显然， $\propto$  具有传递性，即若  $A \propto B$ ， $B \propto C$ ，则  $A \propto C$ 。

## 11.3.2 NP完全问题类

### 1. NP完全问题的定义

【NP 难】称问题  $X$  是 NP 难的当且仅当对任何  $X' \in NP$  都有  $X' \propto X$ 。

【NP 完全】称问题  $X$  是 NP 完全的当且仅当

(1)  $X \in NP$  ; (2)  $X$  是 NP 难的

【NPC】所有 NP 完全问题构成的类称为 NP 完全问题类，记为 NPC。

【注】并非所有 NP 难问题都是 NP 完全的，例如，确定性算法的停机问题是 NP 难的，却不是 NP 的（请参阅《算法概论》）。

### 2. NP完全问题的存在性

定理3（Cook定理）给出了第一个NP完全问题。该定理是Cook于1971年提出并证明的，Cook也因此获得Turing奖。

【定理3】布尔表达式的可满足性问题是NP完全的，即  $SAT \in NPC$ 。

【证明】参阅Ellis Horowitz等的《Computer Algorithms, C++》。

### 3. NP完全问题的一个性质

【定理4】设  $X$  是 NP 完全的，则  $X \in P$  当且仅当  $P = NP$ 。

【证明】只需证明当  $X \in P$  时  $NP \subseteq P$  成立。对任何  $Q \in NP$ ，由  $X \in NPC$  可知， $Q \propto X$ 。由  $X \in P$  和  $Q \propto X$  可知， $Q \in P$ 。

定理4表明 NP 完全问题有一种令人惊奇的性质，如果一个 NP 完全问题能够在多项式时间内成功求解，那么每一个 NP 问题都可以在多项式时间内成功求解，即  $P = NP$ 。但是，目前还没有一个 NP 完全问题有多项式时间算法。因此，大多数科学家猜测  $P \neq NP$ 。

### 4. 几种复杂性类的关系

如果  $P \neq NP$ ，则前述几种复杂性类的关系可能如图11-4所示。

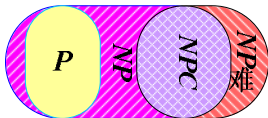


图11-4 几种复杂性类的关系

## 11.4 一些典型的NP完全问题

### 11.4.1 NP完全性的证明方法

【定理5】设  $X$  是一个已知的 NP 完全问题。若一个新问题  $X' \in NP$ ，且  $X \propto X'$ ，则  $X'$  是 NP 完全的。

【证明】对任何  $Q \in NP$ ，由  $X \in NPC$  可知， $Q \propto X$ 。因为  $X \propto X'$ ，所以  $Q \propto X'$ 。由  $Q \propto X'$  和  $X' \in NP$  可知， $X' \in NPC$ 。

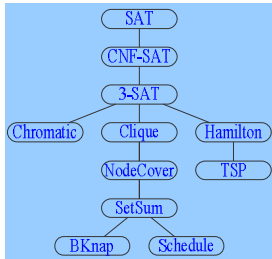
由定理5可知，要证明一个问题  $X'$  是 NP 完全的，只需完成下列两个步骤。

- (1)  $X'$  是 NP 的。为  $X'$  寻找一个多项式时间内成功求解的非确定性算法。
- (2)  $X'$  是 NP 难的。利用一个已知的 NP 完全问题  $X$ ，证明  $X \propto X'$ 。

## 11.4.2 NP完全树

利用上述证明方法，人们很快证明了许多问题的NP完全性。例如，Cook证明了  $SAT \in NPC$  的第二年（1972），Karp提出了21个重要的NP完全问题（后来Karp也获得Turing奖，名字NP也是Karp给出的）。到1979年发现的NP完全问题就有约300种，现在已经发现了几千个NP完全问题。

这些NP完全问题都是直接或间接地以SAT的NP完全性为基础而得到证明的。由此逐渐生长出一棵以SAT为根的NP完全问题树。如图11-5所示的NP完全树是整个NP完全树的一小部分。每个结点代表一个NP完全问题，该问题可在多项式时间内变换为它的任一儿子结点表示的问题。



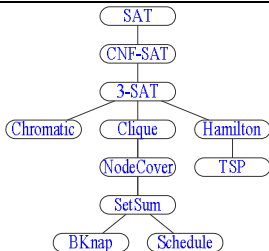


图11-5 NP完全树

下面分类介绍图11-5所示的NP完全树包含的NP完全问题。

### 11.4.3 满足性问题

#### 1. 布尔表达式的可满足性问题SAT

给定一个布尔表达式，判定它是否可满足。

#### 2. 合取范式的可满足性问题CNF-SAT

**【含义】**如果一个布尔表达式是一些子句的合取（And运算），则称为合取范式，简称为CNF。其中，子句是一些文字的析取（Or运算），文字是变量  $x$  或变量的否  $\bar{x}$ 。例如  $E_1 = (x_1 + x_2)(x_2 + x_3)(\bar{x}_1 + \bar{x}_2 + x_3)$  是一个合取范式，而  $E_2 = x_1x_2 + x_3$  不是合取范式（是析取范式）。当  $x_1 = x_3 = 0$  而  $x_2 = 1$  时， $E_1 = 1$ ，所以  $E_1$  是可满足的。

**【问题】**给定一个合取范式，判定它是否可满足。

**【注】**实际上，因为Cook定理的证明过程稍加修改后可以证明合取范式的可满足性问题是NP完全的，所以通常将  $\text{CNF-SAT} \in \text{NPC}$  当作Cook定理的另一种形式。详情请参阅Ellis Horowitz等的《Computer Algorithms, C++》。

### 3. 三元合取范式的可满足性问题3-SAT

【含义】三元合取范式的每个子句恰好是3个文字的析取。例如， $E = (x_1 + x_2 + x_5)(x_3 + x_4 + \bar{x}_5)$  是一个三元合取范式。当  $x_1 = x_2 = x_3 = x_4 = x_5 = 1$  时， $E = 1$ ，所以  $E$  是可满足的。

【问题】给定一个三元合取范式，判定它是否可满足。通常以该问题为基础证明其他问题的NP完全性。



## 11.4.4 图问题

### 1. 团问题Clique

给定一个 $n$ 阶无向图 $G$ 和一个正整数 $k$ ，判定 $G$ 是否存在 $k$ 团（或顶点数不小于 $k$ 的团）。

### 2. 顶点覆盖问题NodeCover

给定一个 $n$ 阶无向图 $G$ 和一个正整数 $k$ ，判定 $G$ 是否存在 $k$ 顶点覆盖（或顶点数不大于 $k$ 的顶点覆盖）。

### 3. 哈密尔顿回路问题Hamilton

一个无向图是否含有哈密尔顿回路。

### 4. 旅行商问题TSP

给定一个边权值非负的无向加权图 $G$ 和一个非负数 $t$ ，判定 $G$ 中是否存在费用为 $t$ （或不超过 $t$ ）的旅行。

## 5. 着色问题Chromatic

给定一个无向图 $G$ 和 $m$ 种颜色, 判断图 $G$ 是否可 $m$ 着色。

实际上, 3-着色问题 (一个无向图是否可3着色) 也是 $NP$ 完全的。

## 11.4.5 子集问题与调度问题

### 1. 子集和问题SetSum

给定正整数集合 $S$ 和一个正整数 $t$ ，判定 $S$ 是否存在和为 $t$ 的子集。

### 2. 0-1背包问题BKnap

给定正数 $c$ 和 $v$ ，及正数数组 $\{v_i\}_{i=0}^{n-1}$ 和 $\{w_i\}_{i=0}^{n-1}$ ，判定是否存在满足 $\sum\{w_i x_i\}_{i=0}^{n-1} \leq c$ 的0/1数组 $\{x_i\}_{i=0}^{n-1}$ ，使得 $\sum\{v_i x_i\}_{i=0}^{n-1} = v$  (或 $\sum\{v_i x_i\}_{i=0}^{n-1} \geq v$ )。

### 3. 多机调度问题Schedule

【问题】已知 $n$ 个独立作业各自需要的处理时间，判断是否存在一种调度方案使这些作业都能够在规定的时间 $t$ 内由 $m$ 台相同的机器处理完毕。

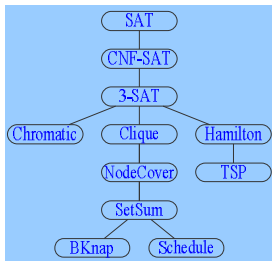
实际上，2机调度问题（即 $m=2$ ）也是NP完全的。

## 11.5 NP完全性的证明举例

前一节已经说明，要证明一个问题  $X'$  是  $NP$  完全的，只需完成下列两个步骤。

- $X'$  是  $NP$  的。为  $X'$  寻找一个多项式时间内成功求解的非确定性算法。
- $X'$  是  $NP$  难的。利用一个已知的  $NP$  完全问题  $X$ ，证明  $X \propto X'$ 。

这里以图11-5所示  $NP$  完全树中从3-SAT开始的中间分枝中包含的问题为例演示一些证明新问题的  $NP$  完全性的一些常用手段。



## 11.5.1 3-SAT\*

【问题】给定一个三元合取范式，判定它是否可满足。

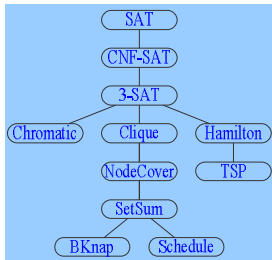
【定理6】3-SAT是  $NP$  完全的。

【证明】

(1) 前面给出了多项式时间内成功求解SAT的非确定性算法，该非确定性算法也可以在多项式时间内成功求解3-SAT，所以3-SAT是  $NP$  的。

(2) 证明CNF-SAT可在多项式时间内变换为3-SAT。

设  $S$  是一个合取范式， $C = x_1 + x_2 + \dots + x_k$  为  $S$  的一个子句。将  $C$  变换为三元合取范式  $C'$ ，并证明  $C=1$  当且仅当  $C'=1$ 。分下列4种情况讨论。



① 当  $k \geq 4$  时，取

$$C' = (x_1 + x_2 + y_1)(x_3 + \bar{y}_1 + y_2) \dots (x_i + \bar{y}_{i-2} + y_{i-1}) \dots \\ (x_{k-2} + \bar{y}_{k-4} + y_{k-3})(x_{k-1} + x_k + \bar{y}_{k-3})$$

其中  $y_1, y_2, \dots, y_{k-3}$  是新加入的变量。

( $\Rightarrow$ ) 若  $C=1$ ，必有某一  $x_i$  的值为 1，指定  $y_1, y_2, \dots, y_{i-2}$  的值为 1， $y_{i-1}, y_i, \dots, y_{k-3}$  的值为 0，显然，此时  $C'=1$ 。

( $\Leftarrow$ ) 设  $C'=1$ 。因为当所有  $x_i=0$  时，

$$y_1(\bar{y}_1 + y_2) \dots (\bar{y}_{i-2} + y_{i-1}) \dots (\bar{y}_{k-4} + y_{k-3}) \bar{y}_{k-3} = 1$$

将导致  $y_1, y_2, \dots, y_{k-3}$  和  $\bar{y}_{k-3}$  均为 1，矛盾。所以，必有某一  $x_i=1$ ，此时，

$$C = x_1 + x_2 + \dots + x_k = 1。$$

② 当  $k=3$  时，取  $C'=C$ 。显然， $C=1$  当且仅当  $C'=1$ 。

③ 当  $k=2$  时，取  $C'=(x_1+x_2+z)(x_1+x_2+\bar{z})$ 。

( $\Rightarrow$ ) 若  $C=x_1+x_2=1$ ，则无论  $z$  取何值，都有  $C'=1$ 。

( $\Leftarrow$ ) 若  $C'=1$ ，则必有  $C=x_1+x_2=1$ ，否则，将有  $C'=0$ 。

④ 当  $k=1$  时，取  $C' = (x_1 + y + z)(x_1 + \bar{y} + z)(x_1 + y + \bar{z})(x_1 + \bar{y} + \bar{z})$ 。

( $\Rightarrow$ ) 若  $C = x_1 = 1$ ，则显然  $C' = 1$ 。

( $\Leftarrow$ ) 若  $C' = 1$ ，则  $C = x_1 = 1$ 。否则，

$$\begin{aligned} C' &= (y + z)(\bar{y} + z)(y + \bar{z})(\bar{y} + \bar{z}) \\ &= (y\bar{y} + yz + z\bar{y} + zz)(y\bar{y} + y\bar{z} + \bar{y}z + \bar{z}z) \\ &= (yz + \bar{y}z + z)(y\bar{z} + \bar{y}z + \bar{z}) \\ &= z\bar{z} \\ &= 0 \end{aligned}$$

矛盾。

由①~④可知，CNF-SAT可在  $O(|S|)$  时间内变换为3-SAT，故3-SAT是  $NP$  完全的。这里， $|S|$  表示合取范式  $S$  的长度。

## 11.5.2 团问题

【问题】给定一个 $n$ 阶无向图 $G$ 和一个不小于 $n/3$ 的正整数 $k$ ，判定 $G$ 是否存在 $k$ 团（或顶点数不小于 $k$ 的团）。

【定理7】团问题是  $NP$  完全的。

【证明】

(1) 容易看出，下列非确定性算法可在 $O(n^2)$ 时间内成功求解团问题。

```
void Clique(const Matrix<bool> &G, int k, vector<bool> &X)
{
    int n = G.rows(), m = 0; // m为选取的顶点数
    for(int i = 0; i < n; ++i) // 非确定性地选取候选解, O(n)
        X[i] = Choice(0, 1), m += X[i];
    if(m < k) Failure();
    for(int i = 0; i < n; ++i) // 确定性地检查候选解是否是团, O(n^2)
        for(int j = i + 1; j < n; ++j)
            if(X[i] == 1 and X[j] == 1 and G(i, j) != 1) Failure();
    Success();
}
```



(2) 证明三元合取范式可满足性问题可在多项式时间内变换为团问题。

设  $f = C_1 C_2 \dots C_m$  是一个三元合取范式，其中  $C_i = v_{i,1} + v_{i,2} + v_{i,3}$ ， $1 \leq i \leq m$ 。构造图  $G = (V, E)$ ，其中

$$V = \{v_{1,1}, v_{1,2}, \dots, v_{m,2}, v_{m,3}\}, E = \{(v_{r,i}, v_{s,j}) \mid r \neq s, v_{r,i} \neq \bar{v}_{s,j}\}$$

变换实例如图11-6所示。

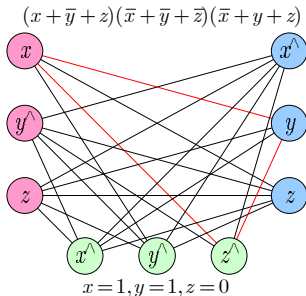
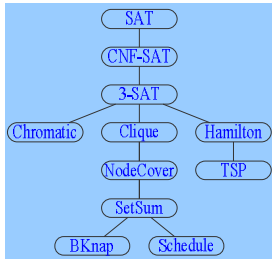


图11-6 三元合取范式变换成图



易知，该构造过程可在  $O(m^2)$  时间内完成。

下面证明  $f$  是可满足的当且仅当  $G$  有  $m$  团。

( $\Rightarrow$ ) 若  $f=1$ ，则  $C_i = v_{i,1} + v_{i,2} + v_{i,3} = 1$  ( $1 \leq i \leq m$ )，从而  $v_{i,1}, v_{i,2}, v_{i,3}$  至少有一个值为1，从中任取一个，记作  $v_{i,f(i)}$ 。令  $V' = \{v_{1,f(1)}, \dots, v_{m,f(m)}\}$ 。显然， $|V'| = m$ 。因为对任何  $v_{i,f(i)}, v_{j,f(j)} \in V'$ ，都有  $i \neq j$ ，由  $v_{i,f(i)} = v_{j,f(j)} = 1$  可知  $v_{i,f(i)} \neq \bar{v}_{j,f(j)}$ ，所以  $(v_{i,f(i)}, v_{j,f(j)}) \in E$ 。这说明  $V'$  是  $G$  的一个  $m$  团。

( $\Leftarrow$ ) 若  $V'$  是  $G$  的一个  $m$  团。因为对任何  $1 \leq i \leq m$ ， $v_{i,1}, v_{i,2}, v_{i,3}$  之间没有边相连，不妨设  $V' = \{v_{1,f(1)}, \dots, v_{m,f(m)}\}$ 。根据  $G$  的构造，对任何  $r \neq s$ ，都有  $v_{r,f(r)} \neq \bar{v}_{s,f(s)}$ ，所以，可以取  $v_{1,f(1)} = \dots = v_{m,f(m)} = 1$ 。显然，此时  $f=1$ 。

### 11.5.3 顶点覆盖问题

【问题】给定一个 $n$ 阶无向图 $G$ 和一个正整数 $k$ ，判定 $G$ 是否存在 $k$ 顶点覆盖（或顶点数不大于 $k$ 的顶点覆盖）。

【定理8】顶点覆盖问题是 $NP$ 完全的。

【证明】

(1) 容易看出，下列非确定性算法可在 $O(n^2)$ 时间内成功求解顶点覆盖问题。

```
void NodeCover(const Matrix<bool> &G, int k, vector<bool> &X)
{
    int n = G.rows(), m = 0; // m为选取的顶点数
    for(int i = 0; i < n; ++i) // 非确定性地选取候选解X, O(n)
        X[i] = Choice(0, 1), m += X[i];
    if(m > k) Failure();
    for(int i = 0; i < n; ++i) // 确定性地检查X是否为顶点覆盖, O(n^2)
        for(int j = i + 1; j < n; ++j)
            if(G(i, j) == 1 and X[i] != 1 and X[j] != 1) Failure();
    Success();
}
```

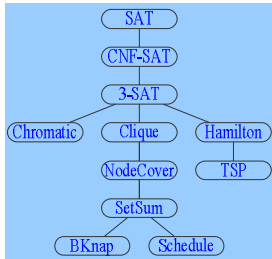
(2) 证明团问题可多项式时间变换为顶点覆盖问题。

易知，可以在  $O(n^2)$  时间内构造出图  $G$  的补图  $\bar{G}$ 。

下面证明  $G$  有  $n-k$  团当且仅当  $\bar{G}$  有  $k$  顶点覆盖。

( $\Rightarrow$ ) 设  $V'$  是  $G$  的  $n-k$  团。显然， $|V - V'| = k$ 。对  $\bar{G}$  的任何边  $(u, v)$ ，都有  $(u, v) \notin E$ 。因为  $V'$  是  $G$  的团，所以  $u \notin V'$  或  $v \notin V'$ ，即  $u \in V - V'$  或  $v \in V - V'$ ，从而边  $(u, v)$  被  $V - V'$  覆盖。因此， $V - V'$  是  $\bar{G}$  的大小为  $k$  的顶点覆盖。

( $\Leftarrow$ ) 设  $V'$  是  $\bar{G}$  的  $k$  顶点覆盖。显然， $|V - V'| = n - k$ 。对任何  $u, v \in V - V'$ ，因为  $u \notin V'$  且  $v \notin V'$ ，所以  $(u, v)$  不是  $\bar{G}$  的边，从而  $(u, v)$  是  $G$  的边。这说明， $V - V'$  是  $G$  的  $n-k$  团。



## 11.5.4 子集和问题\*

【问题】给定正整数集合 $S$ 和一个正整数 $t$ ，判定 $S$ 是否存在和为 $t$ 的子集。

【定理9】子集和问题是 $NP$ 完全的。

【证明】

(1) 容易看出，下列非确定性算法可在 $O(n)$ 时间内成功求解子集和问题。

```
void SetSum(const vector<int> &S, int t, vector<bool> &X)
{
    int n = S.size();
    int w = 0; // 子集和
    for(int i = 0; i < n; ++i) // 选取候选解, 计算和,  $O(n)$ 
        if(Choice(0, 1) == 1) X[i] = 1, w += S[i];
        else X[i] = 0;
    if(w == t) Success();
    else Failure();
}
```

(2) 证明顶点覆盖问题可在多项式时间内变换为子集和问题。

设  $G=(V,E)$  是一个无向图，其中  $V=\{v_0,v_1,\dots,v_{n-1}\}$ ， $E=\{e_0,e_1,\dots,e_{e-1}\}$ 。使用修正的4进制数构造正整数集合  $S$  和正整数  $t$ （图11-7给出了一个由顶点覆盖实例构造子集和实例的例子）。

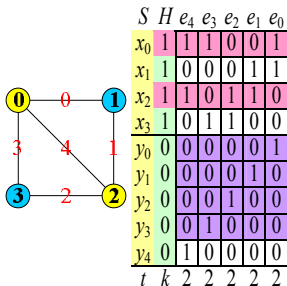
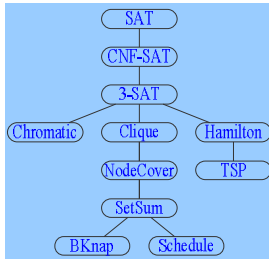


图11-7 由顶点覆盖构造子集和



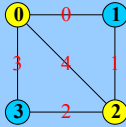
令  $S = X \cup Y$ ，其中， $X = \{x_0, x_1, \dots, x_{n-1}\}$ ， $x_i = 4^e + \sum \{x_{ij} 4^j\}_{j=0}^{e-1}$ ，  
 $x_{ij} = \text{iif}(v_i \text{ cover } e_j, 1, 0)$ ， $Y = \{y_0, y_1, \dots, y_{e-1}\}$ ， $y_i = 4^i$ ， $y_{ij} = \text{iif}(i = j, 1, 0)$ 。  
 对给定的正整数  $k$ ，令  $t = k4^e + \sum \{2 * 4^j\}_{j=0}^{e-1}$ 。

显然，该构造过程可在  $O(e^2)$  时间内完成。

下面证明  $G$  有  $k$  顶点覆盖当且仅当  $S$  有和为  $t$  的子集。

首先，由  $S$  的构造可知，对任何  $0 \leq j < e$ ，

都有  $\sum \{x_{ij}\}_{i=0}^{n-1} + \sum \{y_{ij}\}_{i=0}^{e-1} = 3$  (由每条边正好有两个端点可知  $\sum \{x_{ij}\}_{i=0}^{n-1} = 2$ )，也就是说使用修正的4进制数对  $S$  计算子集和时，第0到  $e-1$  位都不超过3，肯定不会有进位。

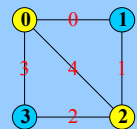


	$S$	$H$	$e_4$	$e_3$	$e_2$	$e_1$	$e_0$
$x_0$	1	1	1	0	0	1	
$x_1$	1	0	0	0	1	1	
$x_2$	1	1	0	1	1	0	
$x_3$	1	0	1	1	0	0	
$y_0$	0	0	0	0	0	1	
$y_1$	0	0	0	0	1	0	
$y_2$	0	0	0	1	0	0	
$y_3$	0	0	1	0	0	0	
$y_4$	0	1	0	0	0	0	
$t$	$k$	2	2	2	2	2	

( $\Rightarrow$ ) 设  $V'$  是  $G$  的一个  $k$  顶点覆盖，令  $X' = \{x_i : v_i \in V'\}$ ， $Y' = \{y_j : \sum \{x_{ij} : v_i \in V'\} = 1\}$ 。易知， $X' \cup Y'$  是  $S$  的一个和为  $t$  的子集。

( $\Leftarrow$ ) 设  $S'$  是  $S$  的一个和为  $t$  的子集，令  $X' = X \cap S'$ ， $Y' = Y \cap S'$ ， $V' = \{v_i : x_i \in X'\}$ 。由  $t = k4^e + \sum \{2 * 4^j\}_{j=0}^{e-1}$  和  $(y_i \text{ idiv } 4^e) = 0$  可知  $|V'| = |X'| = (t \text{ idiv } 4^e) = k$ 。

对任何  $e_j \in E$ ，由  $S$  的构造方法和  $t = k4^e + \sum \{2 * 4^j\}_{j=0}^{e-1}$  可知  $\sum \{x_{ij} : x_i \in X'\} + \sum \{y_{ij} : y_i \in Y'\} = 2$ ，而  $\sum \{y_{ij} : y_i \in Y'\} \leq 1$ ，所以  $\sum \{x_{ij} : x_i \in X'\} \geq 1$ ，从而在  $\{x_{ij} : x_i \in S'\}$  中必定有某个  $x_{ij} = 1$ ，也就是说在  $V'$  中必定有某个顶点  $v_i$  覆盖了边  $e_j$ 。根据顶点覆盖的定义， $V'$  是  $G$  的  $k$  顶点覆盖。



	$S$	$H$	$e_4$	$e_3$	$e_2$	$e_1$	$e_0$
$x_0$	1	1	1	0	0	1	
$x_1$	1	0	0	0	1	1	
$x_2$	1	1	0	1	1	0	
$x_3$	1	0	1	1	0	0	
$y_0$	0	0	0	0	0	1	
$y_1$	0	0	0	0	1	0	
$y_2$	0	0	0	1	0	0	
$y_3$	0	0	1	0	0	0	
$y_4$	0	1	0	0	0	0	
$t$	$k$	2	2	2	2	2	2



## 11.5.5 0-1背包问题

【问题】给定正数  $c$  和  $v$ , 及正数数组  $V = \{v_i\}_{i=0}^{n-1}$  和  $W = \{w_i\}_{i=0}^{n-1}$ , 判定是否存在满足  $\sum \{w_i x_i : 0 \leq i < n\} \leq c$  的0/1数组  $\{x_i\}_{i=0}^{n-1}$ , 使得  $\sum \{v_i x_i\}_{i=0}^{n-1} = v$  (或  $\sum \{v_i x_i\}_{i=0}^{n-1} \geq v$ )。

【定理10】0-1背包问题是  $NP$  完全的。

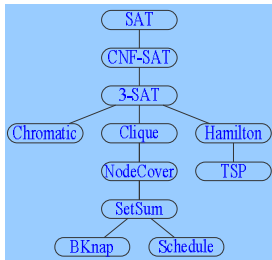
【证明】

(1) 容易看出, 下列非确定性算法可在  $O(n)$  时间内成功求解0-1背包问题。

```
void Knap(const vector<double> &V, const vector<double> &W,  
          double c, double t, vector<bool> &X)  
{ // 效益数组V, 重量数组W, 背包容量c  
  int n = min(V.size(), W.size()); // 物品数  
  double fv = 0, fw = 0; // 最后效益, 最后重量  
  for(int i = 0; i < n; ++i) // 选取候选解, 计算效益和重量  
    if(Choice(0, 1) == 1) X[i] = 1, fv += V[i], fw += W[i];  
    else X[i] = 0;  
  if(fw <= c and fv >= t) Success();  
  else Failure();  
}
```

(2) 证明子集和问题可在多项式时间内变换为0-1背包问题。

对于子集和问题的实例  $\langle S, t \rangle$ ，其中  $S$  是正整数集合， $t$  是一个正整数，构造0-1背包问题的实例  $\langle V, W, c, v \rangle = \langle S, S, t, t \rangle$ 。显然， $\langle V, W, c, v \rangle$  的构造能在  $O(|S|)$  时间内完成。根据  $\langle V, W, c, v \rangle$  的构造易知， $\sum \{s_i x_i\}_{i=0}^{n-1} = t$  当且仅当  $\sum \{v_i x_i\}_{i=0}^{n-1} = v$  与  $\sum \{w_i x_i\}_{i=0}^{n-1} \leq c$  都成立。



## 11.5.6 多机调度问题

【问题】已知  $n$  个独立作业各自需要的处理时间，判断是否存在一种调度方案使这些作业都能够在规定的时间  $t$  内由  $m$  台相同的机器处理完毕。

【定理11】多机调度问题是  $NP$  完全的。

【证明】

(1) 容易看出，下列非确定性算法可在  $O(n)$  时间内成功求解多机调度问题。

```
void Schedule(const vector<int> &J, int m, int t, vector<int> &X)
{
    int n = J.size();
    vector<int> T(m, 0); // 各机器处理时间
    // 非确定性地为每个作业选取机器,  $O(n)$ 
    for(int i = 0; i < n; ++i)
        X[i] = Choice(0, m - 1), T[X[i]] += J[i];
    // 确定性地计算处理时间,  $O(n)$ 
    if(*max_element(begin(T), end(T)) <= t) Success();
    else Failure();
}
```

(2) 证明子集和问题可在多项式时间内变换为多机调度问题。

设  $S = \{x_1, x_2, \dots, x_n\}$  是一个正整数集合， $t$  是一个正整数。由此构造一个多机调度问题实例，该实例包含  $n+2$  项独立作业  $\{j_1, j_2, \dots, j_n, j_{n+1}, j_{n+2}\}$ ，处理时间分别是  $\{x_1, x_2, \dots, x_n, 1+t, \sum S+1-t\}$ ，由2台相同的机器处理。显然，该构造能在  $O(n)$  时间内完成。

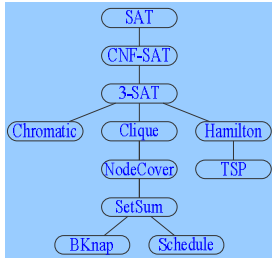
下面证明当且仅当  $S$  有和为  $t$  的子集时，这些作业能在  $\sum S+1$  时间内由2台机器处理完毕。

( $\Rightarrow$ ) 设  $S$  有一和为  $t$  的子集  $S'$ 。如果将  $\{j_i : x_i \in S'\} \cup \{j_{n+2}\}$  的作业安排到机器1，其余作业安排到机器2，则这些作业能在  $\sum S+1$  时间内由2台机器处理完毕。

( $\Leftarrow$ ) 设这些作业都能在  $\sum S+1$  时间内由2台机器处理完毕，且作业  $j_{n+2}$  安排到机器  $k$ 。若安排到机器  $k$  的作业集是  $J$ ，则  $S$  的子集  $S' = \{x_i : j_i \in J - \{j_{n+2}\}\}$  是  $S$  的一个和为  $t$  的子集。

由该证明过程可得下列结论。

【推论】2机调度问题是  $NP$  完全的。



## 11.6 近似算法与 $P = NP$

以旅行商问题的近似算法为例说明。

【问题】给定一个边权值非负的无向图 $G$ ，从 $G$ 中找出一个费用最小的旅行。

### 11.6.1 旅行商问题的近似算法

#### 1. 算法思路

对于给定的无向加权图，可以利用最小生成树算法（这里选用Prim算法）设计寻找近似最优旅行的算法。

- (1) 在图中选取一个起始顶点。
- (2) 用Prim算法找出一棵以起始顶点为根的最小生成树。
- (3) 先根遍历最小生成树，得到一个顶点表。
- (4) 将该顶点表对应的回路做为近似最优的旅行。

【注】要使用该方法，必须保证给定的无向加权图是一个完全图，即每一对顶点都对应一条权值不是无穷大的边。

## 2. 举例说明

图11-8说明了上述TSP近似算法的运行过程及结果。

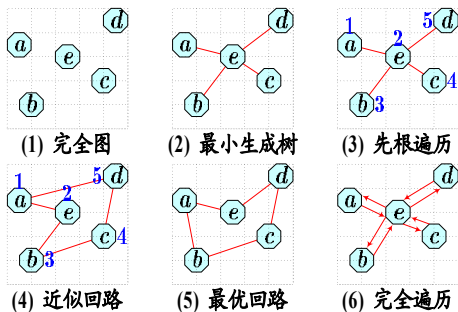
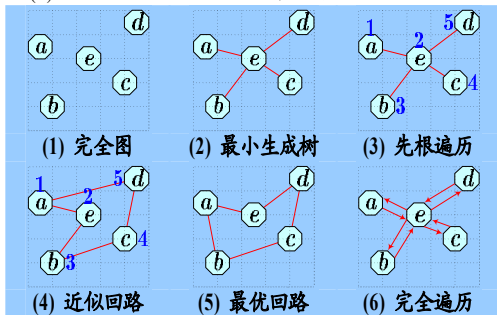


图11-8 TSP近似算法的运行过程及结果

图11-8(1)所示的完全图是一个平面图，边权值就是顶点间的几何距离，该图的一个最优旅行为 $abcdea$ 。如果起始顶点选用 $a$ ，则遍历顺序如图11-8(3)所示，得到如图11-8(4)所示的近似回路 $aebcda$ 。

容易看出，如果起始顶点选用 $e$ ，则得到如图11-8(2)所示的根为 $e$ 的最小生成树，由此得到如图11-8(5)所示的回路 $eabcde$ ，等价于回路 $abcde a$ ，正好是最优旅行。





### 3. 算法描述

为方便其他示例调用, 将这里的算法描述保存在文件TSP.h中。

(1) 先根遍历最小生成树。PreOrder()用于先根遍历最小生成树。

```
#pragma once
#include "Prim.h" // 使用Prim算法获得生成树

void PreOrder(const vector<int> &Prev, int v, vector<int> &X)
{ // 先根遍历生成树Prev(数组Prev保存各顶点的双亲), 得到顶点表X
  int n = Prev.size(); // 顶点数
  X.push_back(v); // 访问根v(v加入顶点表)
  for(int w = 0; w < n; ++w) // 遍历子树
    if(w != v and Prev[w] == v) // v是w的双亲
      PreOrder(Prev, w, X); // 访问子树
} // 耗时O(n^2)
```

## (2) 主算法。

```
auto TSP(const Matrix<double> &G, int v) // 邻接矩阵, 起始顶点
{
    vector<int> Prev, X; // 最小生成树, 结果顶点表
    Prim(G, v, Prev); // 使用Prim算法获得最小生成树Prev, 耗时 $O(n^2)$ 
    PreOrder(Prev, v, X); // 先根遍历树Prev, 得到顶点表X, 耗时 $O(n^2)$ 
    return X; // 顶点表X对应的回路就是近似最优旅行
} // 耗时 $O(n^2)$ 
```

## 4. 复杂性分析

如果完全图的顶点数为 $n$ ，则生成最小生成树的计算时间为 $O(n^2)$ ，遍历最小生成树的计算时间为 $O(n^2)$ ，其余计算时间 $O(1)$ 。所以，上述算法的计算时间为 $O(n^2)$ 。

## 5. 多次执行以获得更优解

由上述示例的分析可知, 如果通过选取不同的起始顶点多次执行该算法, 可能获得比一次执行更优的旅行。

```
#include "TSP.h" // 一次执行, 耗时 $O(n^2)$ 

auto TSP(const Matrix<double> &G) // 邻接矩阵
{
    int n = G.rows(); // 顶点数
    vector<int> BX; // 最优旅行
    double BC = inf; // 最优耗费, 初始值为无穷大

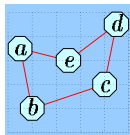
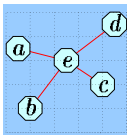
    for(int v = 0; v < n; ++v) // 执行n次
    {
        auto X = TSP(G, v); // 当前旅行(以v为起始顶点)
        double CC = 0; // 当前耗费, 初始值为0
        for(int i = 0; i < n; ++i) // 计算当前耗费
            CC += G(X[i % n], X[(i + 1) % n]);
        if(CC < BC) // 当前旅行更优
            BX = X, BC = CC; // 修改结果
    }

    return BX;
}
```

6. 测试<sup>☆</sup>

使用如图11-8(1)所示的完全图，该图的最优旅行为(0, 1, 2, 3, 4, 0)。

```
int main()
{ Matrix<double> G = // 邻接矩阵
  { {0, 5.1, 7.6, 8.2, 4.1},
    {5.1, 0, 6.3, 9.9, 5},
    {7.6, 6.3, 0, 5.1, 3.6},
    {8.2, 9.9, 5.1, 0, 5},
    {4.1, 5, 3.6, 5, 0}
  };
  cout << TSP(G, 0) << endl; // 0 4 1 2 3 (28.7)
  cout << TSP(G, 4) << endl; // 4 0 1 2 3 (25.6, optimal)
  cout << TSP(G) << endl; // 3 4 0 1 2 (25.6, optimal)
}
```



## 11.6.2 性能分析

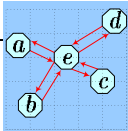
### 1. 满足三角不等式的情况

从实际应用中抽象出来的旅行商问题通常具有一些特殊性质。比如，费用函数  $c$  往往具有三角不等式性质，即对任何 3 个顶点  $u, v, w \in V$ ，都有  $c(u, w) \leq c(u, v) + c(v, w)$ 。当图  $G$  中的顶点就是平面上的点，任意 2 顶点间的费用就是这 2 点间的欧氏距离时，费用函数  $c$  就满足三角不等式性质。

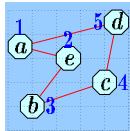
当费用函数满足三角不等式时，可以证明该算法的性能比不大于 2。换句话说，若用  $H^*$  表示图  $G$  的最优旅行，用  $H$  表示该算法得到的旅行，则  $C(H) \leq 2C(H^*)$ ，其中  $C(A) = \sum \{c(u, v) : (u, v) \in A\}$ 。

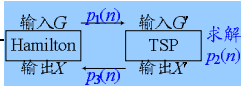
下面证明这一结论。

设  $T$  是图  $G$  的一棵最小生成树， $T'$  是  $H^*$  去掉一条边后形成的一棵生成树，则  $C(T) \leq C(T') \leq C(H^*)$ 。从  $T$  的根出发，绕着  $T$  的外缘逆时针遍历  $T$  再回到根，可以得到  $G$  的一条回路  $W$ （这种遍历称为  $T$  的完全遍历，在如图11-8所示的示例中， $W = aebecedea$ ）。因为  $W$  经过  $T$  的每一条边恰好2次，所以  $C(W) = 2C(T) \leq 2C(H^*)$ 。



另外，近似回路  $H$  的顶点序列正好是  $W$  的顶点序列按顺序去掉重复出现的顶点（末尾顶点除外）以后保留下来的顶点（在如图11-8所示的示例中， $W = aebecedea$ ， $H = aebcda$ ）。由费用函数的三角不等式性质可知  $C(H) \leq C(W)$ ，从而  $C(H) \leq 2C(H^*)$ 。





## 2. 一般情况

在一般情况下，当  $P \neq NP$  时，对任何  $\rho > 1$ ，旅行商问题都不存在性能比为  $\rho$  的多项式时间近似算法。也就是说，若存在某个  $\rho > 1$ ，使得旅行商问题有一个性能比为  $\rho$  的多项式时间近似算法  $A$ ，则  $P = NP$ 。

**【证明】**通过证明可以由算法  $A$  获得Hamilton回路问题的一个多项式时间算法来证明  $P = NP$ 。

(1) 输入变换。对无向图  $G = (V, E)$ ，令  $n = |V|$ ，构造加权图  $G' = (V', E')$ ，其中  $V' = V$ ， $E' = \{(u, v) \mid u, v \in V', u \neq v\}$ ，且费用  $w(u, v) = \text{iif}((u, v) \in E, 1, \rho n)$ 。显然，该构造过程可在多项式时间内完成，且最优旅行的费用  $t^* \geq n$ 。

(2) 输出变换。假设用算法  $A$  求解  $G'$  获得的费用为  $t$ 。下面证明  $G$  有Hamilton回路当且仅当  $t = n$ 。

若  $G$  有Hamilton回路，则该回路在  $G'$  中对应一个费用为  $n$  的旅行，即  $t^* = n$ ，从而  $t \leq \rho t^* = \rho n$ 。此时， $t = n$ 。理由是  $t \neq n$  会导致  $t \geq (n-1) + \rho n > \rho n$ 。

若  $t = n$ ，则费用为  $t$  的旅行对应  $G$  的一条Hamilton回路。

由算法  $A$  和上述变换可知，依次执行输入变换、算法  $A$  和输出变换就是Hamilton回路问题的一个多项式时间算法。

## 11.7 练习题

### 11.7.1 基础训练题

1. 解释下列概念。  
 $P$  类问题,  $NP$  类问题,  $NPC$  类问题,  $NP$  难类问题
2. 将下列问题改写成判定问题, 并证明这些判定问题是  $NP$  的。  
3-SAT, 团问题, 顶点覆盖问题, 哈密尔顿回路问题, 旅行商问题, 着色问题, 子集和问题, 0-1背包问题, 多机调度问题。
3. 证明下列结论。
  - (1) CNF-SAT可在多项式时间内变换为3-SAT。
  - (2) 3-SAT可在多项式时间内变换为团问题。
  - (3) 团问题可在多项式时间内变换为顶点覆盖问题。
  - (4) 顶点覆盖问题可在多项式时间内变换为团问题。
  - (5) 子集和问题可在多项式时间内变换为0-1背包问题。
  - (6) 子集和问题问题可在多项式时间内变换为多机调度问题。
  - (7) Hamilton回路问题可在多项式时间内变换为旅行商问题。



4. 证明下列判定问题是  $NPC$  的。

3-SAT, 团问题, 顶点覆盖问题, 子集和问题, 0-1背包问题, 多机调度问题。

5. 已知子集和问题是  $NP$  完全的, 证明均衡划分问题 (将一个正整数集分为和相等的两个子集) 是  $NP$  完全的。

6. 已知Hamilton回路问题是  $NP$  完全的, 证明旅行商问题是  $NP$  完全的。

7. 若有限集  $X$  的一个子集族  $F$  满足  $\cup\{S: S \in F\} = X$ , 则称子集族  $F$  是  $X$  的一个集合覆盖或  $F$  覆盖  $X$ 。已知顶点覆盖问题  $NP$  完全的, 证明集合覆盖问题 (有限集  $X$  的集合覆盖  $F$  中是否有子集数为  $k$  的子集族能够覆盖  $X$ ) 是  $NP$  完全的。

8. 请使用一种程序设计语言为旅行商问题构造一个近似算法。

## 11.7.2 实验题

1. 将求最优旅行的一种近似算法改写成蒙特卡洛算法, 以达到在多项式时间内

### 11.7.3 进阶训练题

1. SAT的NP完全性。已知3-着色问题（一个无向图是否有3-着色）是NP完全的，证明SAT是NP完全的。

2. 2-SAT的P性。2-SAT是每个子句正好2个文字的合取范式的可满足性问题。证明2-SAT是P类问题，并给出一种求解该问题的高效算法（最坏情形耗时以 $n^3$ 为上界）

3. Hamilton回路问题的NP完全性。已知3-SAT是NP完全的，证明无向图的哈密尔顿回路问题是NP完全的。

4. 顶点覆盖问题的NP完全性。已知3-SAT是NP完全的，证明顶点覆盖问题是NP完全的。

5. 子集和问题的NP完全性。已知精确覆盖问题是NP完全的，证明子集和问题是NP完全的。

6. 0-1背包问题的NP完全性。已知均衡划分问题是NP完全的，证明0-1背包问题是NP完全的。

7. 精确覆盖问题的NP完全性。若子集族 $F$ 是有限集 $X$ 的一个的集合覆盖，则 $F$ 中覆盖 $X$ 的互不相交的子集族 $T$ 称为 $\langle X, F \rangle$ 的精确覆盖。已知3-SAT是NP完全的，证明精确覆盖问题（ $\langle X, F \rangle$ 是否有精确覆盖）是NP完全的。

8. 团问题的NP完全性。已知顶点覆盖问题是NP完全的，证明团问题是NP完全的。

9. 均衡划分问题的NP完全性。已知子集和问题是NP完全的，证明均衡划分问题（一个整数集是否可以正好划分为2个和相等的子集）是NP完全的。

10. 多机调度问题的NP完全性。已知均衡划分问题是NP完全的，证明多机调度问题是NP完全的。

11. 着色问题的NP完全性。已知3-SAT是NP完全的，证明着色问题是NP完全的。