《算法设计与分析》

第十章 近似算法

马丙鹏

2023年12月04日

第十章 近似算法

- 10.1 概述
- 10.2 图问题中的近似算法
- 10.3 组合问题中的近似算法

10.1 概述

■ 1. 设计思想

- 近似算法是求解 NP 类问题的一种有效策略。
- 许多NP类问题实质上是最优化问题，即要求在满足约束条件的前提下，使某个目标函数达到极值（极大或极小）的最优解。
- 现实世界中，很多问题的输入数据是用测量方法获得的，而测量的数据本身就存在着一定程度的误差，这类问题允许最优解有一定程度的误差，近似最优解常常能满足实际问题的需要。
- 近似算法的基本思想是用近似最优解代替最优解，以换取算法设计上的简化和时间复杂性的降低。
- 近似算法找到的可能不是一个最优解，但一定会为待求解的问题提供一个解。

10.1 概述

■ 1. 设计思想

- □如何衡量近似最优解的质量呢？
  - ➢假设近似算法求解最优化问题，且每个可行解对应的目标函数值均为正数。
  - ➢若一个最优化问题的最优值为c^*，求解该问题的近似算法求得的近似最优值为 c，则将该近似算法的近似比η 定义为：

$$\eta = \max\left(\frac{c}{c^*}, \frac{c^*}{c}\right)$$

  - ➢用相对误差λ 表示近似算法的近似程度，定义为：

$$\lambda = \left|\frac{c - c^*}{c^*}\right|$$

10.1 概述

■ 2. 求π的近似值

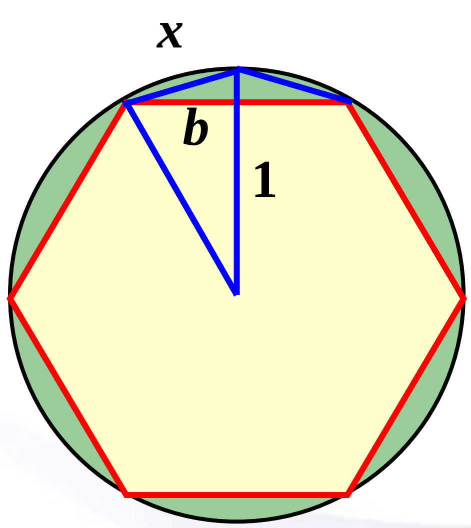
- 问题描述
  - 请用正多边形逼近法求 π 的近似值。
- 求解思路
  - 用圆内接正多边形的边长和半径之间的关系，不断将边数翻倍并求出边长，
  - 重复这一过程，正多边形的周长就逐渐逼近圆的周长，
  - 只要圆内接正多边形的边数足够多，就可以求得所需精度的π值。

10.1 概述

■ 2. 求π的近似值

□求解思路

➢简单起见，设单位圆的半径是 1，则单位圆的周长是 2×π，设单位圆内接正 *i* 边形的边长为 2*b*，边数加倍后正 2*i* 边形的边长为 *x*，则：



$$x=\sqrt{b^2+(1-\sqrt{1-b^2})^2}=\sqrt{2-2\sqrt{1-b^2}}$$

$$2\mathrm{i}*x=2\pi$$

10.1 概述

■ 2. 求π的近似值

□算法实现

➢因为圆的内接正 6 边形的边长等于半径，所以从内接正 6 边形开始。程序如下：

参数 e 表示精度要求，设变量 b 表示正多边形翻倍之前边长的一半，变量 x 表示翻倍之后的边长，

```
double Pi(double e) {
    int i = 6;
    double b, x = 1;   //正6边形的边长为1
    do {
        b = x / 2;                        // 原先边长的一半
        i = i * 2;                        // 新的边数
        x = sqrt(2 - 2 * sqrt(1.0 - b*b));  // 新的边长
    } while (i * x - i * b > e);
    cout<<"圆的内接多边形的边数是："<<i<<endl;
    return (i * x)/2;      // pi *2 = i * x;
}
```

第十章 近似算法

- 10.1 概述
- 10.2 图问题中的近似算法
- 10.3 组合问题中的近似算法

10.2 图问题中的近似算法

■ 1. 顶点覆盖问题

□问题描述

- ➢无向图 $G=(V, E)$的顶点覆盖是求顶点集 V 的一个子集 V'，使得若(u, v)是 G 的一条边，则 $v\in V'$ 或 $u\in V'$。
- ➢顶点覆盖问题是求图 G 的最小顶点覆盖，即含有顶点数最少的顶点覆盖。

□求解思路

- ➢顶点覆盖问题是一个 NP 难问题，目前尚未找到一个多项式时间算法。
- ➢可以采用如下策略找到一个近似最小顶点覆盖：

10.2 图问题中的近似算法

■ 1. 顶点覆盖问题

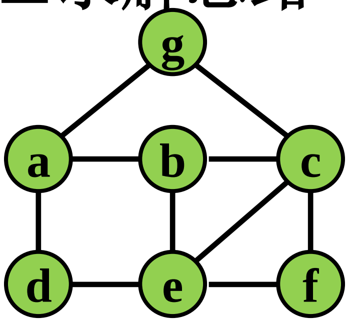
□求解思路

- ➢初始时边集$E'=E$，顶点集 $V'=\{\}$，每次从边集 E' 中任取一条边(u, v)，
- ➢把顶点u和v加入顶点集 V'，再把与顶点 u和 v相邻接的所有边从边集 E'中删除，
- ➢重复上述过程，直到边集 E' 为空，最后得到的顶点集 V' 是无向图的一个顶点覆盖。
- ➢由于每次把尽量多的相邻边从边集 E' 中删除，可以期望 V' 中的顶点数尽量少，
- ➢但不能保证 V' 中的顶点数最少。

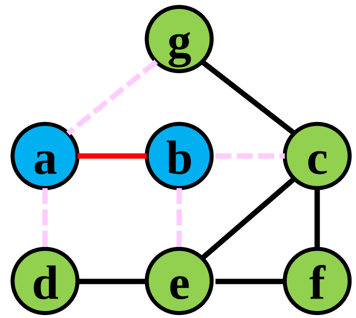
10.2 图问题中的近似算法

■ 1. 顶点覆盖问题

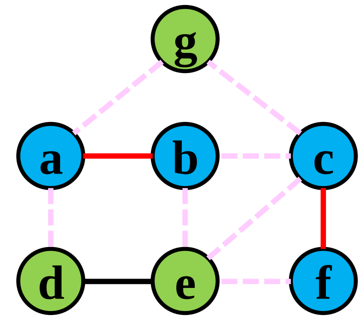
□求解思路



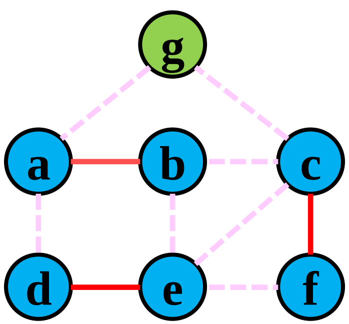
(a) 一个无向图



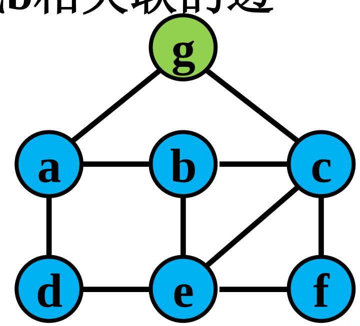
(b) v’={a, b} 删除与a或b相关联的边



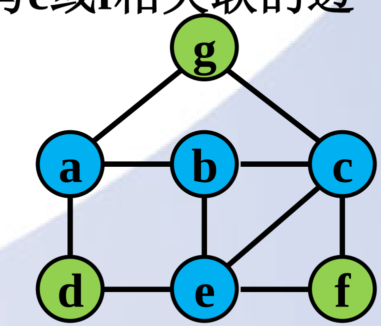
(c) v’={a, b, c, f} 删除与c或f相关联的边



(d) v’={a, b, c, f, d, e} 删除与d或e相关联的边



(e) 近似最小顶点覆盖 v’={a, b, c, f, d, e}



(e) 最小顶点覆盖 v’={a, b, c, e}

10.2 图问题中的近似算法

■ 1. 顶点覆盖问题

□算法实现

- ➢设数组x[n] 表示集合 V'，x[i]=1 表示顶点 i 在集合 V' 中，算法如下：

算法：顶点覆盖问题的近似算法VertexCover

输入：无向图$G=(V, E)$

输出：覆盖顶点集合x[n]

1. 初始化：x[n] = {0}; $E' = E$；
2. 循环直到 E' 为空
   2.1 从 E' 中任取一条边(u, v);
   2.2 将顶点 u 和 v 加入顶点覆盖中：x[u] = 1; x[v] = 1;
   2.3 从 E' 中删去与 u 和 v 相关联的所有边;

10.2 图问题中的近似算法

- **1. 顶点覆盖问题**
  - 算法分析
    - 设无向图 *G* 采用邻接表存储，由于算法对每条边只进行一次删除操作，设图 *G* 含有 *n* 个顶点 *e* 条边，时间复杂性为O(n+e)。
    - 算法VertexCover的近似比(略)
      - ✓设 *A* 表示在步骤 2.1 中选取的边的集合，近似算法求得的近似最小顶点覆盖为 *V'*，图 *G* 的最小顶点覆盖为*V**，
      - ✓由于图 *G* 的任一顶点覆盖一定包含 *A* 中各边的至少一个顶点，因此：

$$|A|\le|V^*|$$

10.2 图问题中的近似算法

■ 1. 顶点覆盖问题

□算法分析

- ➢算法VertexCover的近似比。
  - ✓因为近似算法选取了一条边，在将其顶点加入顶点覆盖后，将 E' 中与该边的两个顶点相关联的所有边从 E' 中删除，因此，下一次再选取的边与该边没有公共顶点。
  - ✓由数学归纳法易知，A 中的所有边均没有公共顶点。算法结束时，顶点覆盖中的顶点数 $|V'|=2|A|$。由此可得：
    $|V'|\le 2|V^*|$
  - ✓即算法VertexCover的近似比为2。

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

10.2 图问题中的近似算法

■ 2. TSP问题

□问题描述

- 设无向图 $G=(V, E)$的顶点在一个平面上，边$(i, j)\in E$ 的代价均为非负整数，且两个顶点之间的距离为欧几里得距离。
- TSP问题是求图 G 的最短哈密顿回路。

□求解思路

- 设 c_{ij} 表示边$(i, j)\in E$ 的代价，由于顶点之间的距离为欧氏距离，对于图 G 的任意 3 个顶点 i、j 和 k，满足三角不等式：$c_{ij}+c_{jk}\geq c_{ik}$ 。
- 可以证明，满足三角不等式的 TSP问题仍为 NP难问题。

10.2 图问题中的近似算法

- **2. TSP问题**
  - **求解思路**
    - 求解 TSP问题的近似算法：

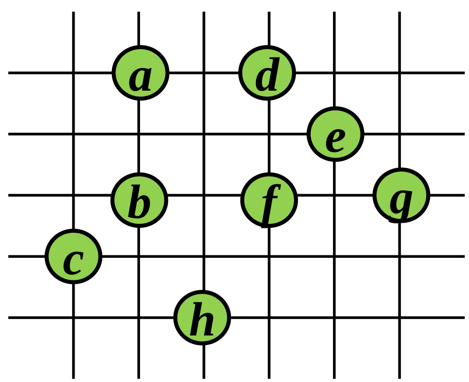
      ① 采用 Prim算法生成图的最小生成树 T；

      ② 对 T 进行深度优先遍历，得到遍历序列 L；

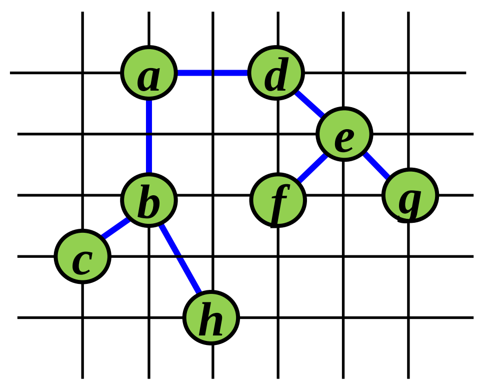
      ③ 由序列 L 构成的回路就是哈密顿回路。

10.2 图问题中的近似算法

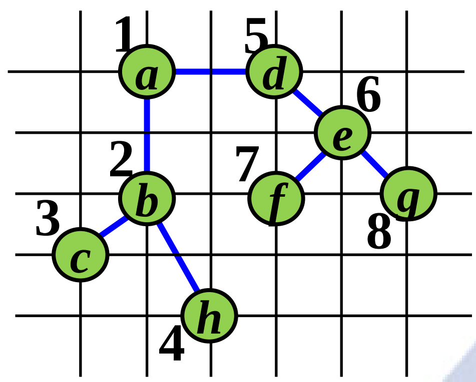
■ 2. TSP问题



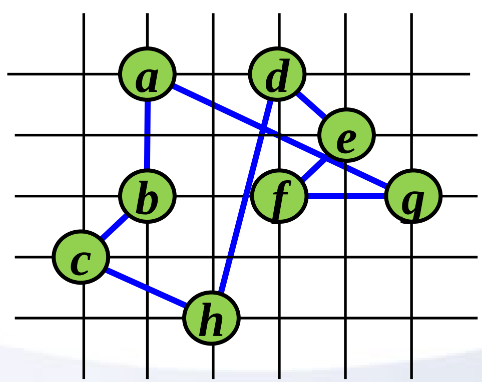
(a) 无向图*G*



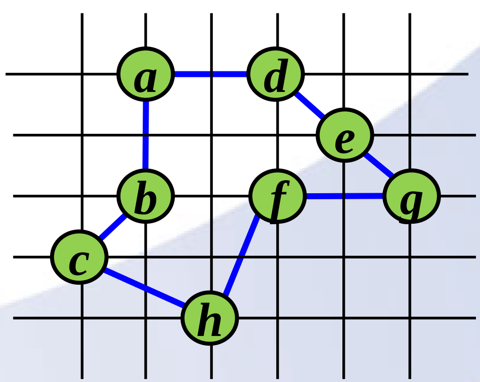
(b) 生成最小生成树*T*



(c) 对*T*进行深度优先遍历



(d) 由遍历序列产生哈密顿回路



(e) TSP问题的最优解

10.2 图问题中的近似算法

■ 2. TSP问题

□算法实现

- ➢设无向带权图 *G* 满足三角不等式，采用代价矩阵存储图 *G*，算法如下：

算法：TSP问题的近似算法TSP

输入：无向带权图*G*=(*V*, *E*)

输出：近似最短回路

1. 在图中任选一个顶点 *v*；
2. 采用Prim算法生成以顶点 *v* 为根结点的最小生成树 *T*
3. 对生成树 *T* 从顶点 *v* 出发进行深度优先遍历，得到遍历序列*L*；
4. 根据 *L* 得到哈密顿回路，返回路径长度；

10.2 图问题中的近似算法

■ 2. TSP问题

□算法分析

- 步骤 2 采用Prim算法构造最小生成树，时间开销是O(n^2)。
- 步骤 3 对对生成树 T 进行深度优先遍历，时间开销是O(n)，
- 因此，时间复杂度为O(n^2)。
- (略) 设无向图 G 的最短哈密顿回路为 H^*，W(H^*)是 H^* 的代价之和；T 是由 Prim算法求得的最小生成树，W(T) 是 T 的代价之和；H 是由近似算法得到的近似解，W(H) 是 H 的代价之和。则有 W(T)≤W(H^*)

10.2 图问题中的近似算法

- **2. TSP问题**
  - 算法分析
    - 设深度优先生成树 T 得到的路线为 R，则有：$W(R)=2W(T)$
    - 近似解 H 是 R 删除了若干中间点得到的，每删除一个顶点恰好是用三角形的一条边取代另外两条边。
    - 由三角不等式可知，这种取代不会增加总代价，所以，有 $W(H)\le W(R)$
    - 从而 $W(H)\le 2W(H^*)$，由此，算法 TSP的近似比为2。

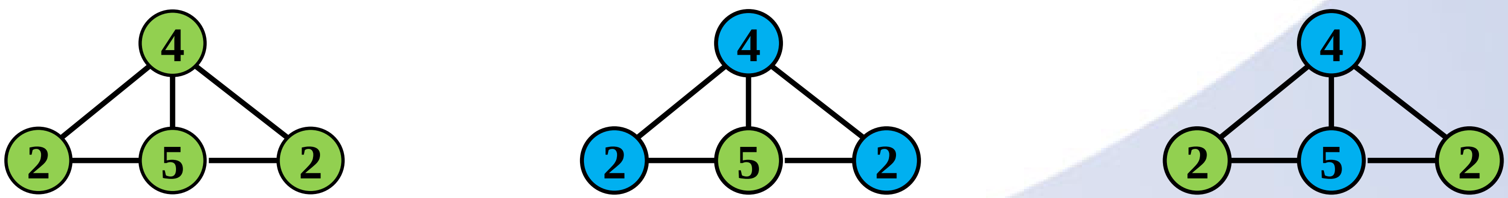
10.2 图问题中的近似算法

■ 3. 带权顶点覆盖问题

□ 问题描述

➢ 对于无向图 *G*=(*V*, *E*)，每个顶点 *v*∈*V* 都有一个权值 w(*v*)，对于顶点集 *V* 的一个子集 *V*′，若(*u*, *v*)∈*E*，则 *v*∈*V*′ 或 *u*∈*V*′，称集合 *V*′ 是图 G 一个的顶点覆盖。

➢ 在图 *G* 的所有顶点覆盖中，所含顶点权值之和最小的顶点覆盖称为 *G* 的最小权值顶点覆盖。



(a) 一个无向图

(b) 权值和为8的顶点覆盖

(c) 权值和为9的顶点覆盖

10.2 图问题中的近似算法

■ 3. 带权顶点覆盖问题

□求解思路

- ➢最小权值顶点覆盖的近似算法采用定价法，不断寻找紧致的顶点加入顶点覆盖集合。
- ➢设 c_{ij} 表示边(i, j)上的权值，w_i 表示顶点 i 的权值，S_i 表示与顶点 i 相关联的所有边的集合，
- ➢定价法要求与顶点 i 所关联的所有边的权值之和必须小于等于该顶点的权值，
- ➢即对于每个顶点 $i \in E$，满足：

$$\sum_{(i,j)\in S_i} c_{ij} \le w_i$$

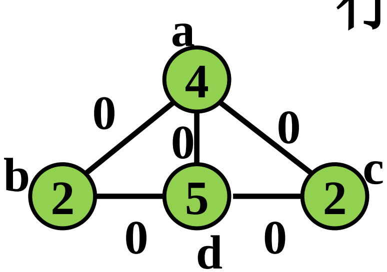
10.2 图问题中的近似算法

■ 3. 带权顶点覆盖问题

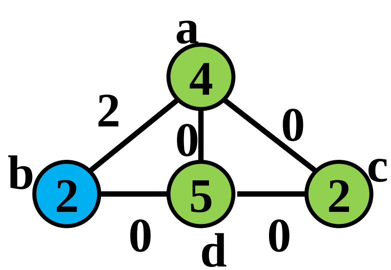
□求解思路

➢如果边(i, j)相关联的两个顶点 i 和 j 均满足上式，则称边(i, j)满足边上权值的公平性，并且将满足的顶点称为紧致顶点。

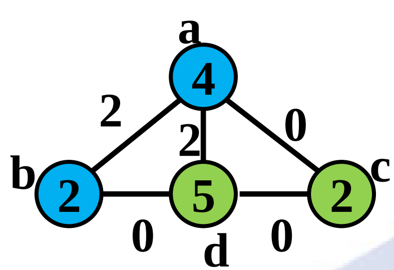
➢定价法对边上权值的设定与寻找覆盖顶点同步进行。



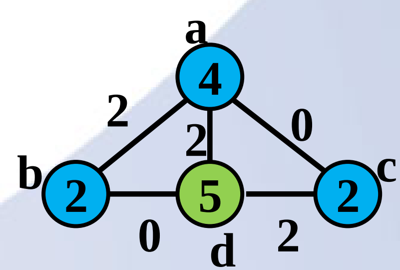
(a) 初始化



(b) 边(a, b)赋权值
顶点b紧致



(c) 边(a, d)赋权值
顶点a紧致



(d) 边(d, c)赋权值
顶点c紧致

10.2 图问题中的近似算法

■ 3. 带权顶点覆盖问题

□算法实现

设图 G 中有 n 个顶点 e 条边，数组w[n]存储顶点的权值，数组p[e]存储边（i, j）的权值，算法如下：

算法：定价法求最小权值顶点覆盖

输入：图 $G=(V, E)$，顶点的权重 w[n]

输出：最小权值顶点覆盖集合 U

1. 将数组 p[e] 初始化为 0；
2. 重复下述操作直到不存在紧致顶点：
   - 2.1 在 E 中选取一条边 (i, j)；
   - 2.2 d1 = 根据顶点 i 计算边 (i, j) 的权值；
   - 2.3 d2 = 根据顶点 j 计算边 (i, j) 的权值；
   - 2.4 如果 d1 < d2，则 U = U + i；p[(i, j)] = d1；
     否则 U = U + j；p[(i, j)] = d2；
3. 返回集合 U；

10.2 图问题中的近似算法

- **3. 带权顶点覆盖问题**
  - 算法分析
    - 步骤 2 每次迭代结束后，至少有一个顶点是紧致的，算法终止时，集合 U 就是一个最小权值顶点覆盖。
    - 对于考察的每一条边，该边依附的两个顶点中有一个顶点是紧致的，所以每一条边会有一个顶点在 U 中，所以 U 是一个顶点覆盖。
    - 定价法是一个 2 倍近似算法

第十章 近似算法

- 10.1 概述
- 10.2 图问题中的近似算法
- 10.3 组合问题中的近似算法

10.3 组合问题中的近似算法

■ 1. 装箱问题

□问题描述

- 设有 n 个物品和若干个容量为 C 的箱子，n 个物品的体积分别为$\{s_1, s_2, \ldots, s_n\}$，且有 $s_i \le C$（$1 \le i \le n$），
- 把所有物品装入箱子，求占用箱子数最少的方案。

□求解思路

- 最优装箱方案通过把 n 个物品划分为若干个子集，每个子集的体积和小于 C，然后取子集个数最少的划分方案。
- 但是，这种划分可能的方案数有 2^n种，在多项式时间内不能够保证找到最优装箱方案。

10.3 组合问题中的近似算法

■ 1. 装箱问题

□求解思路

- ➢采用贪心法设计装箱问题的近似算法，下面是三种贪心策略：

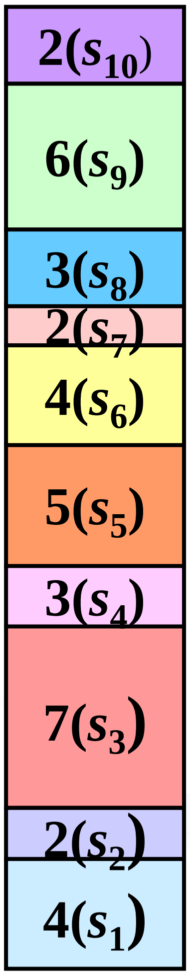
① 首次适宜法。依次取每一个物品，将该物品装入第一个能容纳它的箱子中。

② 最适宜法。依次取每一个物品，将该物品装到目前最满并且能够容纳它的箱子中，使得该箱子装入物品后的闲置空间最小。

③ 首次适宜降序法。将物品按体积从大到小排序，然后用首次适宜法装箱。

10.3 组合问题中的近似算法

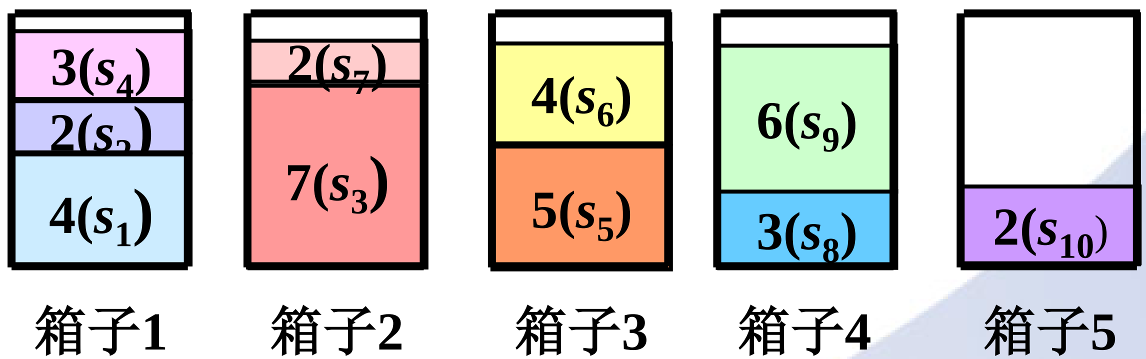
■ 1. 装箱问题



□例

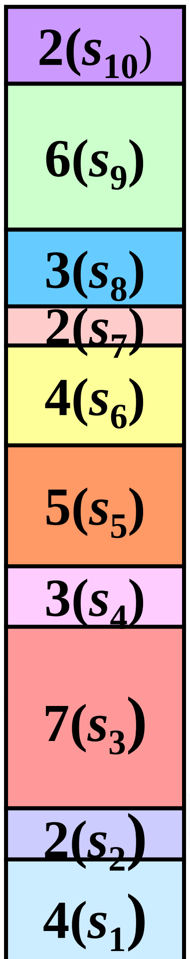
➢有 10 个物品，体积 S＝(4, 2, 7, 3, 5, 4, 2, 3, 6, 2)，若干个容量为 10 的箱子

✓首次适宜法



10.3 组合问题中的近似算法

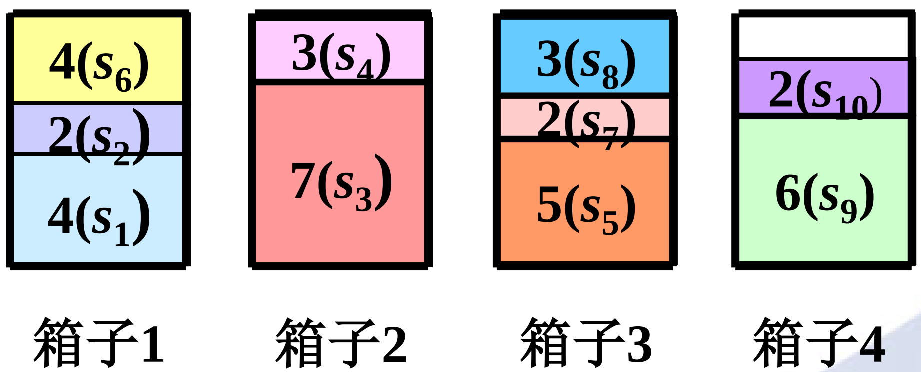
■ 1. 装箱问题



□例

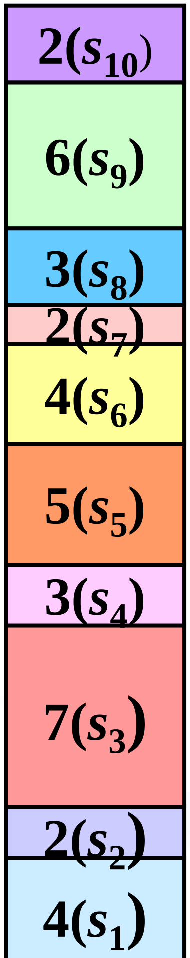
➢有 10 个物品，体积 S＝(4, 2, 7, 3, 5, 4, 2, 3, 6, 2)，若干个容量为 10 的箱子

✓最适宜法



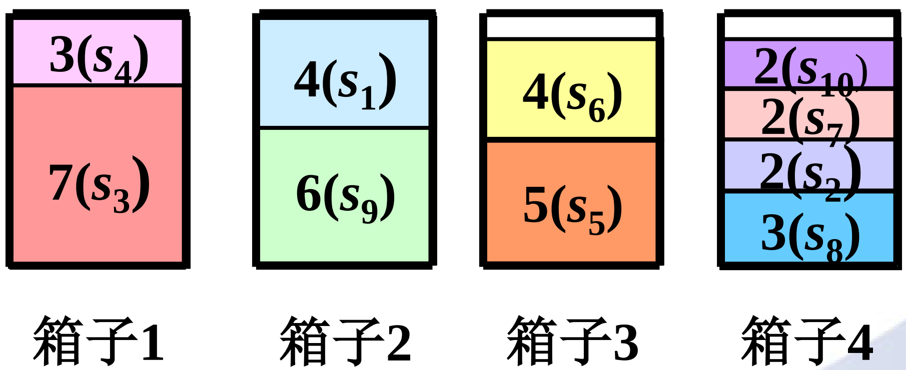
10.3 组合问题中的近似算法

■ 1. 装箱问题



□例

- 有 10 个物品，体积 S＝(4, 2, 7, 3, 5, 4, 2, 3, 6, 2)，若干个容量为 10 的箱子
  - ✓首次适宜降序法



10.3 组合问题中的近似算法

■ 1. 装箱问题

□算法实现

设数组s[n]存储各物品的体积，b[n]存储装入物品的箱子的体积，变量*k*表示装入物品的箱子的最大下标，首次适宜法求解装箱问题的程序如下：

```
int FirstFit(int s[ ], int n, int C, int b[ ])
{
    int i, j, k = -1;
    for (j = 0; j < n; j++)              //所有箱子的体积初始化为0
        b[j] = 0;
    for (i = 0; i < n; i++)             //装入第i个物品
    {
        j = 0;
        while (C - b[j] < s[i])          //查找第1个能容纳物品i的箱子
            j++;
        b[j] = b[j] + s[i];
        if (k < j) k = j;
    }
    return k + 1;                        //返回已装入物品的箱子的个数
}
```

10.3 组合问题中的近似算法

■ 1. 装箱问题

□算法分析

- ➢算法FirstFit的基本语句是查找第 1 个能容纳物品的箱子，对于第 i 个物品，最坏情况下需要试探 i 次，执行次数为

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

- ➢算法FirstFit的近似比(略)
  - ✓假箱子的容量 C 为一个单位的体积，C_i 为第 i 个箱子已装入物品的体积，则 $C_i+C_j>1$（$1 \le i, j \le n$）。

10.3 组合问题中的近似算法

■ 1. 装箱问题

□算法分析

➢算法FirstFit的近似比。

✓设装箱问题的近似解为 **k**，若 **k** 为偶数，则 $C_1+C_2+\ldots+C_k>k/2$，若 **k** 为奇数，则 $C_1+C_2+\ldots+C_k>(k-1)/2$，将两式相加，得：

$$2\sum_{i=1}^{k} C_i > \frac{2k-1}{2}$$

✓设装箱问题的最优解为 **k***，即：$k^* = \sum_{i=1}^{k^*} C_i = \sum_{i=1}^{n} s_i$

✓所以，有：$2\sum_{i=1}^{k} C_i = 2\sum_{i=1}^{n} s_i = 2k^* > \frac{2k-1}{2}$ 由此，算法FirstFit的近似比小于2。

中国科学院大学

University of Chinese Academy of Sciences

10.3 组合问题中的近似算法

■ 2. 多机调度问题

□问题描述

- ➢设有n个作业{1, 2, …, n}，由m台机器{$M_1, M_2, \ldots, M_m$}进行加工处理，作业 i 所需的处理时间为 t_i（$1 \le i \le n$），
- ➢每个作业均可在任何一台机器上加工处理，但不可间断、拆分。
- ➢多机调度问题要求给出一种作业调度方案，使得n个作业在尽可能短的时间由 m台机器加工处理完成

10.3 组合问题中的近似算法

■ 2. 多机调度问题

□求解思路

- ➢可以采用贪心法设计多机调度问题的近似算法，
- ➢贪心策略是最长处理时间作业优先，即把处理时间最长的作业分配给最先空闲的机器，这样可以保证处理时间长的作业优先处理，从而在整体上获得尽可能短的处理时间。

① 当 $m \geq n$时，只要将机器 i的$[0, t_i)$时间区间分配给作业 i 即可；

② 当 $m < n$时，首先将n个作业按其所需的处理时间从大到小排序，然后依此顺序将作业分配给最先空闲的处理机。

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

10.3 组合问题中的近似算法

■ 2. 多机调度问题

□例

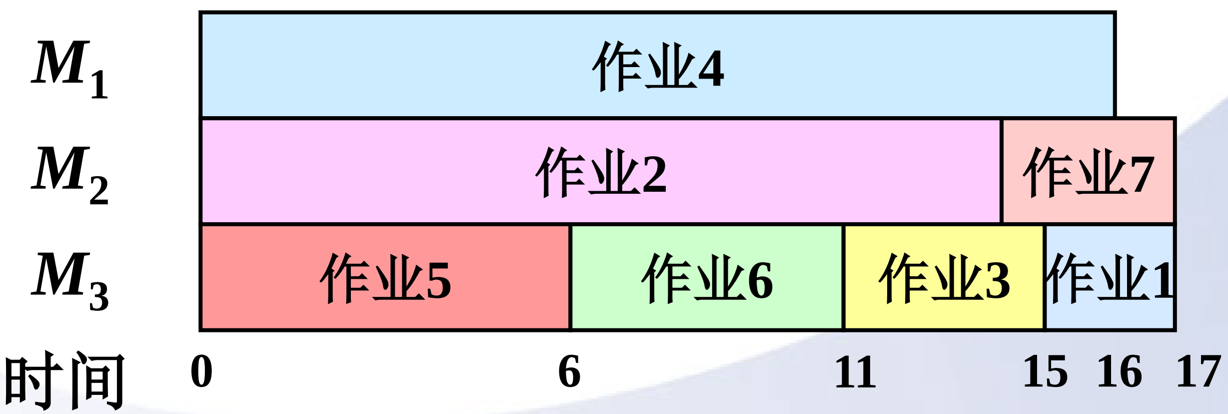
- 设 7 个独立作业{1, 2, 3, 4, 5, 6, 7}由 3 台机器{M_1, M_2, M_3}加工处理，各作业所需的处理时间分别为{2, 14, 4, 16, 6, 5, 3}，
- 首先将这 7 个作业按处理时间从大到小排序，则作业{4, 2, 5, 6, 3, 7, 1}的处理时间为{16, 14, 6, 5, 4, 3, 2}，
- 按照最长处理时间作业优先的贪心策略，近似算法产生的作业调度如下：

10.3 组合问题中的近似算法

■ 2. 多机调度问题

□例

作业	4	2	5	6	3	7	1
时间	16	14	6	5	4	3	2



10.3 组合问题中的近似算法

■ 2. 多机调度问题

□算法实现

设数组t[*n*]存储*n*个作业的处理时间，数组d[*m*]存储*m*台机器的空闲时间，集合数组S[*m*]存储每台机器所处理的作业，算法如下：

算法：多机调度问题MultiMachine

输入：*n*个作业的处理时间t[*n*]，*m*台机器的空闲时间d[*m*]

输出：每台机器所处理的作业 S[*m*]

1．将数组 t[*n*] 由大到小排序，对应的作业序号存储在数组 p[*n*] 中；

2．将数组 d[*m*] 初始化为 0；

3．对前 *m* 个作业进行分配：

  3.1 将第 *i* 个作业分配给第 i 个机器：S[*i*] = {p[*i*]};

  3.2 设置第 *i* 个机器的结束时间：d[*i*] = t[*i*]；

4. 循环变量 *i* 从 *m*~*n*-1，分配作业 *i*：

  4.1 *j* =数组d[*m*]中最小值对应的下标；

  4.2 将作业 *i* 分配给最先空闲的机器 *j*：S[*j*] = S[*j*] + {p[*i*]};

  4.3 机器 *j* 将在d[*j*]后空闲：d[*j*] = d[*j*] + t[*i*];

10.3 组合问题中的近似算法

- **2. 多机调度问题**
  - 算法实现
    - 设二维数组S[m][n]存储每台机器处理的作业，S[i]以队列的形式存储机器i的处理作业，rear[i]为该队列的队尾下标，数组t[n]已排好序，程序如下：

```
void MultiMachine(int t[ ], int n, int d[ ], int m)
{
   int i, j, k, S[m][n], rear[m];
   for (i = 0; i < m; i++)                          //安排前m个作业
   {    S[i][0] = i; rear[i] = 0;  d[i] = t[i]; }//每个作业队列均只有一个作业
   for (i = m; i < n; i++)                        //依次安排余下的每一个作业
      {
         for (j = 0, k = 1; k < m; k++)           //查找最先空闲的机器
            if (d[k] < d[j]) j = k;
         rear[j]++; S[j][rear[j]] = i;             //将作业i插入队列S[j]
         d[j] = d[j] + t[i];
   }
   for (i = 0; i < m; i++)                  //输出每个机器处理的作业
   {
      cout<<"机器"<<i<<"：  ";
      for (j = 0; j <= rear[i]; j++)
          cout<<"作业"<<S[i][j]<<"  ";
      cout<<endl;
   }
}
```

10.3 组合问题中的近似算法

- **2. 多机调度问题**
  - 算法分析
    - 第 1 个循环完成前m个作业的分配，时间开销为 O(m)；
    - 第 2 个循环完成后n-m个作业的分配，时间开销为 O((n-m)×m)，
    - 通常情况下m<<n，则算法的时间复杂度为 O($n\log_2 n$)。

10.3 组合问题中的近似算法

■ 3. 子集和问题

□ 问题描述

- ➢给定一个正整数集合 S={s_1, s_2,…, s_n},
- ➢子集和问题要求在集合 *S* 中，找出不超过正整数 *C* 的最大和。

□ 求解思路

- ➢考虑蛮力法求解子集和问题：

① 将所有子集和的集合初始化为 $L_0=\{0\}$;

② 求子集和包含 s_1的情况，即 L_0的每一个元素加上 s_1(用 L_0+s_1 表示)，则$L_1=L_0\cup(L_0+s_1)=\{0, s_1\}$;

③ 再求子集和包含 s_2 的情况，即 L_1 的每一个元素加上 s_2，则 $L_2=L_1\cup(L_1+s_2)=\{0, s_1, s_2, s_1+s_2\}$;

10.3 组合问题中的近似算法

3. 子集和问题

求解思路

- 依此类推，一般情况下，为求子集和包含 s_i（$1 \le i \le n$）的情况，即 L_{i-1} 的每一个元素加上 s_i，则 $L_i = L_{i-1} \cup (L_{i-1}+s_i)$。
- 因为子集和问题要求不超过正整数 C，所以，每次合并时都要在 L_i 中删除所有大于 C 的元素。

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

10.3 组合问题中的近似算法

3. 子集和问题

例

- S={104, 102, 201, 101}，C=308，求得的最大和是307，相应的子集是{104, 102, 101}。

$L_0=\{0\}$

$L_1=L_0\cup(L_0+104)=\{0\}\cup\{104\}=\{0, 104\}$

$L_2=L_1\cup(L_1+102)=\{0, 104\}\cup\{102, 206\}=\{0, 102, 104, 206\}$

$L_3=L_2\cup(L_2+201)=\{0, 102, 104, 206\}\cup\{201, 303, 305, 407\}$

$=\{0, 102, 104, 201, 206, 303, 305\}$

$L4=L_3\cup(L_2+101)=\{0, 102, 104, 201, 206, 303, 305\}\cup$

$\{101, 203, 205, 302, 307, 404, 406\}$

$=\{0, 101, 102, 104, 201, 203, 205, 206, 302, 303, 305, 307\}$

10.3 组合问题中的近似算法

- 3. 子集和问题
  - 求解思路
    - 在每次合并结束并且删除所有大于 C 的元素后，在不超过近似误差 ε 的前提下，以 $\delta=\varepsilon/n$ 作为修整参数，
    - 对于元素 z，删去满足条件 $(1-\delta)\times y\le z\le y$ 的元素 y，尽可能减少下次参与迭代的元素个数，从而获得算法时间性能的提高。

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

10.3 组合问题中的近似算法

- 3. 子集和问题
  - 算法实现
    - 给定近似参数 ε，算法如下

算法：子集和问题的近似算法SubCollAdd

输入：正整数集合 S，正整数 C，近似参数 ε

输出：最大和

1. 初始化：$L_0=\{0\}$；$\delta=\varepsilon/n$；
2. 循环变量 i 从 $1\sim n$ 依次处理集合 S 中的每一个元素 s_i
   2.1 计算 $L_{i-1}+s_i$；
   2.2 执行合并操作：$L_i=L_{i-1}\cup(L_{i-1}+s_i)$
   2.3 在 L_i 中删去大于 C 的元素；
   2.4 对 L_i 中的每一个元素 z，删去与 z 相差 δ 的元素；
3. 输出 L_n 的最大值；

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

10.3 组合问题中的近似算法

- 3. 子集和问题
  - 例
    - S={104, 102, 201, 101}，C=308，
    - 给定近似参数 ε=0.2，则修整参数为$\delta=\varepsilon/n$=0.05，
    - 最后返回 302 作为子集和问题的近似解，
    - 而最优解为 307。

$L_0=\{0\}$

$L_1= L_0 \cup (L_0 +104)=\{0\}\cup\{104\}=\{0, 104\}$

对L_1进行修整：$L_1=\{0, 104\}$，未删去元素

$L_2= L_1 \cup (L_1+102)=\{0, 104\}\cup\{102, 206\}=\{0, 102, 104, 206\}$

对L_2进行修整：$=\{0, 102, 206\}$，删去元素104

$L_3= L_2 \cup (L_2 +201)=\{0, 102, 206\}\cup\{201, 303, 407\}$

$=\{0, 102, 201, 206, 303\}$

对L_3进行修整：$L_3=\{0, 102, 201, 303\}$，删去元素206

$L_4= L_3 \cup (L_2 +101)=\{0, 102, 201, 303\}\cup\{101, 203, 302, 404\}$

$=\{0, 101, 102, 201, 203, 302, 303\}$

对L_4进行修整：$L_4=\{0, 101, 201, 302\}$，删去元素102、203、303

作业-课后练习29

- 课后练习
  - 给定集合S={3, 7, 5, 9}, C= 20, 近似参数 ε=0.2， 写出近似算法求解子集和问题的过程。

End