

一、算法分析基础知识

记号

$$f(n) = 10n^2 + 5n$$
$$g(n) = 2^n$$

- O : 最坏情况, 函数复杂度小于等于

$$(n) = O(n^2)$$
$$f(n) = O(f(n))$$

- Ω : 最优情况, 复杂度大于等于
- Θ : 精确情况, 复杂度等于

当上界和下界相等时, 用 Θ 来描述算法的复杂度

复杂度分析: 主定理

公式

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), \quad a \geq 1, b > 1$$

1. 若存在 $\epsilon > 0$, $f(n) = O(n^{\log_b(a) - \epsilon})$, 那么 $T(n) = \Theta(n^{\log_b(a)})$
2. 如果 $f(n) = \Theta(n^{\log_b(a)})$, 那么 $T(n) = \Theta(n^{\log_b(a)} \cdot \log n)$
3. 若存在 $\epsilon > 0$, $f(n) = \Omega(n^{\log_b(a) + \epsilon})$, 并且存在常数 $c < 1$ 使得 $n \rightarrow \infty : af\left(\frac{n}{b}\right) \leq cf(n)$, 则 $T(n) = \Theta(f(n))$

分析

- 每个问题会分成 a 个子问题
- 问题可以划分 $\log_b n$ 次, 每次划分都伴随着 $f(n)$
- 对应到递归树中, 最终叶子节点的复杂度是 $\Theta(1)$, 共有 $a^{\log_b n}$ 个叶子节点, 应用换底公式可得 $a^{\log_b n} = n^{\log_b a}$
- 问题可以简化为 $T(n) = pf(n) + n^{\log_b a}$
- 求复杂度等价于比较处理所有叶子节点更耗时还是处理所有划分更耗时
- 第三种情况中为什么多了: 存在常数 $c < 1$ 使得 $n \rightarrow \infty : af\left(\frac{n}{b}\right) \leq cf(n)$
 - 每次划分后 $f(n)$ 并不会保持不变, 而是会跟着划分 $af\left(\frac{n}{b}\right)$
 - $n \rightarrow \infty : af\left(\frac{n}{b}\right) \leq cf(n)$ 要求 $f(n)$ 划分的复杂度不能大于问题本身, 否则没必要分治

参考文献

- [深入浅出理解主定理原理\(Master theorem\)如何计算递归时间复杂度](#)
- [Wiki 主定理](#)

例题: 递归方程

某递归算法的时间复杂度 $T(n)$ 满足递归方程： $T(n) = 2T(n/2) + bn$ ($n \geq 2$) 且 $T(1) = a$ ，其中 $n = 2^k$ ， a, b, k 均为正整数。 $T(n)$ 可用 Θ 表示为

- 展开法

$$\begin{aligned}
 T(n) &= 2T\left(\frac{n}{2}\right) + bn \\
 &= 2\left(2T\left(\frac{n}{4}\right) + \frac{1}{2}bn\right) + bn \\
 &= 4T\left(\frac{n}{4}\right) + 2bn \\
 &= 2^k T\left(\frac{n}{2^k}\right) + kbn \\
 n &= 2^k, k = \log_2 n \\
 T(n) &= 2^k T(1) + kbn \\
 &= an + bn \cdot \log n \\
 &= \Theta(n \log n)
 \end{aligned}$$

- 主定理

$$\begin{aligned}
 a &= 2, b = 2, f(n) = bn \\
 f(n) &= \Theta(n^{\log_b a}) = \Theta(n) \\
 T(n) &= \Theta(n \log n)
 \end{aligned}$$

例题：冒泡排序

算法 RECBUBBLESORT

输入： n 个元素的数组 $A[1 \dots n]$ 。

输出： 按照非降序排列的数组 $A[1 \dots n]$ 。

```

1. sorted ← false
2. sort( $n$ )

```

过程 *sort*(i) //对 $A[1 \dots i]$ 排序

```

1. if  $i \geq 2$  and not sorted then
2.     for  $j \leftarrow 1$  to  $i-1$ 
3.         if  $A[j] > A[j+1]$  then
4.             交换  $A[j]$  与  $A[j+1]$ 
5.             sorted ← false
6.         end if
7.     end for
8.     sort( $i-1$ )                      //递归对  $A[1 \dots i-1]$  排序
9. end if

```

设上述算法中元素比较的最多次数为 $T(n)$ ，则满足递归方程 $T(n) = T(n-1) + n - 1$ ， $T(1) = T(0) = 0$

$$\begin{aligned}
T(n) &= T(n-1) + n - 1 \\
&= T(n-2) + (n-2) + (n-1) \\
&= T(n-3) + (n-3) + (n-2) + (n-1) \\
&= T(n-k) + kn - \frac{k(k+1)}{2} \\
&= T(1) + n(n-1) - \frac{n(n-1)}{2} \\
&= \frac{1}{2}n(n-1) \\
&= \Theta(n^2)
\end{aligned}$$

例题：图的 DFS

假设图 $G = \langle V, E \rangle$ 用邻接矩阵表示， $n = |V|, m = |E|$ 。分析图的深度优先遍历算法 DFS 的时间复杂度。

1. 算法 DFS 中有哪些基本操作？各基本操作耗费时间是多少？

算法 DFS 中基本操作包括：访问结点、访问相邻结点

图中每个结点都恰好访问 1 次，访问结点的总次数为 n ，耗费时间为 $\Theta(n)$ ；

访问相邻结点即扫描邻接矩阵的一行，因此检测总耗费时间为 $\Theta(n^2)$

2. 请根据基本操作的耗费时间，分析得出算法 DFS 的时间复杂度。

算法 DFS 的时间复杂度主要由访问结点的时间、访问相邻结点的时间来决定，其他运算耗费时间不超过这两部的耗费时间。

故算法 DFS 时间复杂度为 $\Theta(n) + \Theta(n^2) = \Theta(n^2)$ 。

二、算法设计基本方法

例题：找到数组中最大的两个元素

假定 $n = 2^k$ (k 为正整数)。设计一个分治算法在数组 $A[1..n]$ 中找最大元素和第二大元素，要求比较次数为 $\frac{3n}{2} - 2$

分析

1. 用文字简要描述递归形式分治算法的基本思路（不用写出完整代码）

1. 当 $n=2$ 时，直接比较即可找到最大值和最小值

2. 当 $n>2$ 时，递归求出 $A[1.. \frac{n}{2}]$ 和 $A[\frac{n}{2}.. n]$ 中的最大元素和次大元素，分别标记为 $[x_1, y_1], [x_2, y_2]$ ，通过比较可以找出最大元素和次大元素

```

1  if x1 > x2:
2      x = x1
3      if x2 > y1:
4          y = x2
5      else:
6          y = y1
7  else:
8      x = x2
9      if y2 > x1:
10         y = y2
11     else:
12         y = x1

```

2. 请给出算法中比较次数 $T(n)$ 满足的递归方程, 并证明 $T(n) = \frac{3n}{2} - 2$

$$T(n) = 2T\left(\frac{n}{2}\right) + 2, T(2) = 1$$

+2 是因为 if 判断需进行两次

- 主定理

$$a = 2, b = 2, f(n) = 2$$

$$n^{\log_b a} = O(f(n))$$

$$T(n) = \Theta(n)$$

啊? 只能给出数量级 🤔

- 展开

$$\begin{aligned} T(n) &= 2^2 T\left(\frac{n}{2^2}\right) + 2 + 2^2 \\ &= 2^k T\left(\frac{n}{2^k}\right) + 2^1 + 2^2 + \dots + 2^k \\ &= 2^k T\left(\frac{n}{2^k}\right) + \frac{2 \cdot (2^k - 1)}{2 - 1} \\ &= 2^{k+1}, k = \log_2 \frac{n}{2} \\ T(n) &= \frac{n}{2} + n - 2 \\ &= \frac{3n}{2} - 2 \end{aligned}$$

代码

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  using namespace std;
5
6  const int INT_MIN = -1;
7
8  struct MaxVal {
9      int max1; // 最大值
10     int max2; // 第二大值
11 };
12
13 MaxVal findMaxTwo(const vector<int>& a, int l, int r) {
14     if (l == r) {
15         return {a[l], INT_MIN};
16     }
17
18     if (r - l == 1) {
19         if (a[l] > a[r]) {
20             return {a[l], a[r]};
21         } else {
22             return {a[r], a[l]};
23         }
24     }
25 }
```

```

26     int m = (l + r) / 2;
27     MaxVal left = findMaxTwo(a, l, m);
28     MaxVal right = findMaxTwo(a, m + 1, r);
29
30     vector<int> candidates = {left.max1, left.max2, right.max1, right.max2};
31     sort(candidates.begin(), candidates.end(), greater<int>()); // 按降序排序
32     return {candidates[0], candidates[1]};
33 }
34
35 int main() {
36     int n;
37     cin >> n;
38     vector<int> a(n);
39     for (int i = 0; i < n; ++i) {
40         cin >> a[i];
41     }
42     MaxVal result = findMaxTwo(a, 0, n - 1);
43     cout << "最大的数是: " << result.max1 << endl;
44     cout << "第二大的数是: " << result.max2 << endl;
45     return 0;
46 }

```

例题：找出数组中最小的 t 个元素

2、考虑在 n 个互不相同元素的数组 $A[1 \dots n]$ 中找出所有前 t 个最小元素的问题（这 t 个元素不要求由小到大有序）。这里 t 不是一个常量，而是作为输入数据的一部分。我们可以很容易用排序算法解决此问题并返回 $A[1 \dots t]$ ，然而耗费时间为 $O(n \log n)$ 。请设计一个时间复杂度较低的算法求解该问题。

（注意，你可以调用选择第 k 小元素算法 $\text{SELECT}(A, \text{low}, \text{high}, k)$ 求解，该算法用于在数组元素 $A[\text{low} \dots \text{high}]$ 中返回第 k 小元素，其时间复杂度为 $\Theta(n)$ 。但不能直接使用 t 次调用算法 SELECT 来解决问题，因为这将耗费时间 $\Theta(tn) = O(n^2)$ （这里 t 不是常量），不符合题意。）

1. 使用 SELECT 函数找到最小元素 x
2. 使用 SELECT 函数找到第 t 小的元素 y
3. 扫描数组 A ，将 $x \leq a[i] \leq y$ 的元素交换到 $A[1 \dots t]$

调用两次 SELECT 的时间复杂度为 $\Theta(n)$ ，扫描数组并交换元素的时间复杂度为 $\Theta(n)$ ，因此算法的时间复杂度为 $\Theta(n)$

例题：Dijkstra

3、(1) 简述 Dijkstra 算法的基本思路。

(2) 在图 1 中应用 Dijkstra 算法能求出从 s 到其它所有结点的最短路径吗？为什么？

(3) 考虑用 Dijkstra 算法求出图 2 中从 s 到其它所有结点的最短路径。请给出主要计算过程及计算结果。

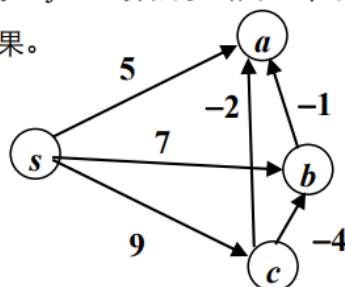


图 1

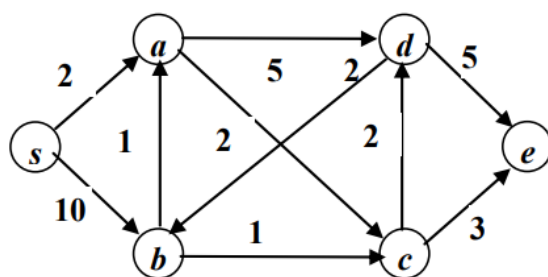


图 2

(1) Dijkstra 的基本思路

1. 准备一个 $dist$ 数组, $dist[i]$ 表示起点到节点 i 的距离
2. 初始化 $dist[start] = 0$, $start$ 为起点编号, $dist[x] = \infty, x \neq start$
3. 查找 $dist$ 中未处理过且距离最小的节点 x , 遍历 x 的所有出边 $\langle x, y \rangle$, y 是未处理过的节点
 1. 若 $dist[y] > dist[x] + |\langle x, y \rangle|$, 则更新距离
 2. 标记 x 已经处理

(2) Dijkstra 处理负权图

不能, 图中存在负权边

由于 $dist[a] < dist[b] < dist[c]$, 且 a 没有出边, 因此 b 的出边比 c 先更新, 也先标记, 而 c 到 b 的边是负权边 $dist[c] + (-4) = 5$, 小于已经标记的 $dist[b] = 7$, 却没有更新

(3) Dijkstra 求最短路径过程

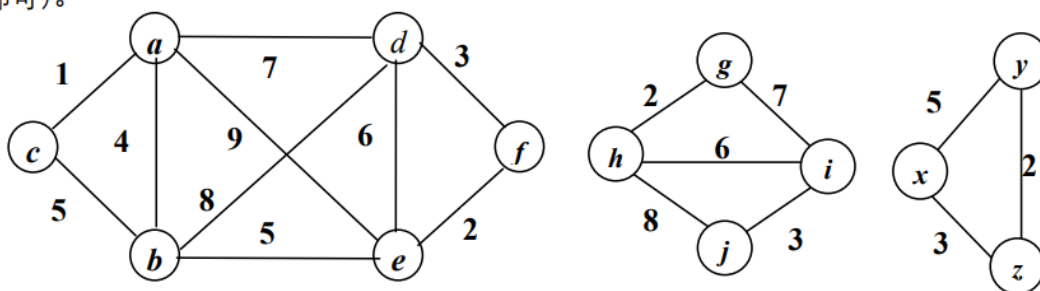
迭代节点	s	a	b	c	d	e
s	0	2	10			
a				4	7	
c					6	7
d			8			
e						
b						
	0	2	8	4	6	7

例题：最小生成树

4、给定带权无向图 G , 如果 G 是连通图, 则 G 的最小成本生成森林就是 G 的最小生成树; 如果 G 不连通, 则 G 的各个连通分支的最小生成树组成图 G 的最小成本生成森林。

(1) 最小生成树算法有 Prim 算法和 Kruskal 算法。这两个算法中哪一个算法更适合求解上述最小成本生成森林? 简述原因。

(2) 带权无向图 G 的图形如下, 试求解其最小成本生成森林 (画出最小成本生成森林即可)。

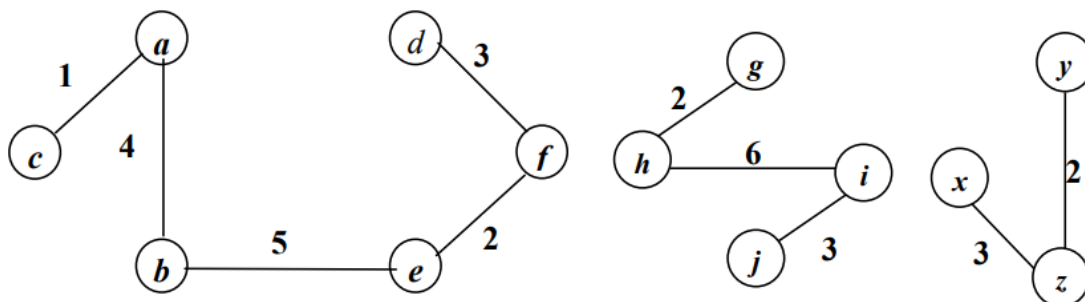


(1) 求解最小成本森林

- 对于不连通图
 - Prim 算法从某点开始依次在该点所在连通分支中加边，最终能得到**该连通分支**的最小生成树
 - Kruskal 算法按照边权由小到大依次考虑边的加入，能求得**所有连通分支**的最小生成树

当图不连通时，需要在每个连通分支上单独运行 Prim 算法来处理。因此 Kruskal 算法更适合求解最小成本生成森林问题

(2) 求最小成本森林



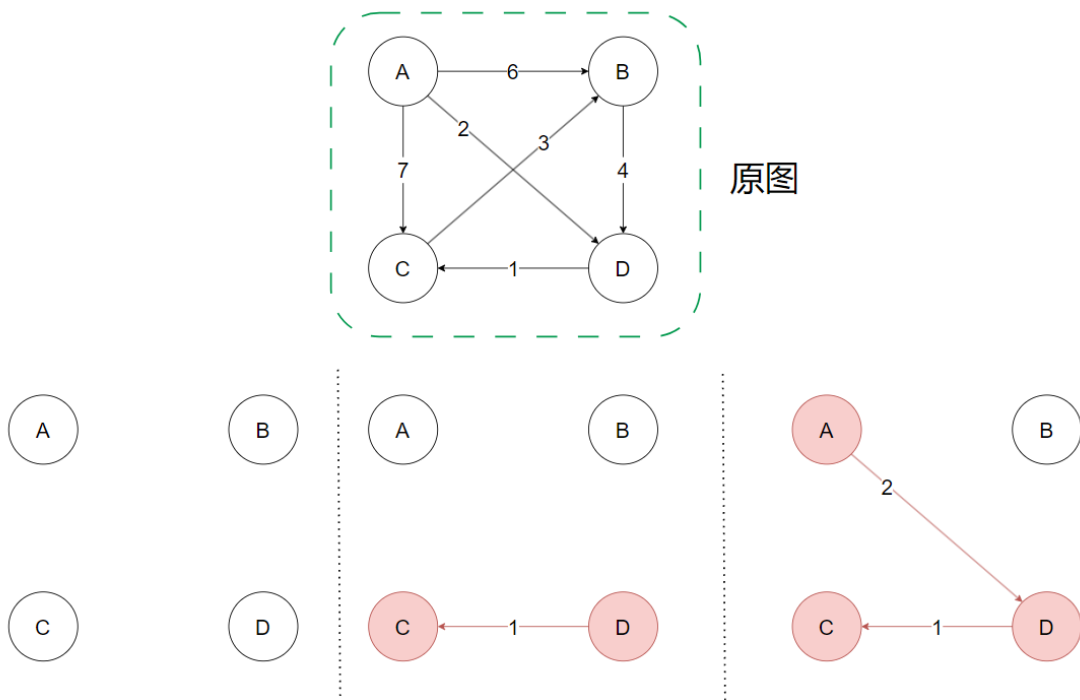
例题：最大间隔聚类

5、【最大间隔聚类问题】给定非负带权无向图 $G=(V,E)$ 及正整数 $k(1 \leq k \leq |V|)$ ，试将 G 划分为 k 个非空子图 G_1, G_2, \dots, G_k 使得任意两子图的最小间距最大。这里两子图的间距是指跨于这两子图之间的最小权的边的权值。

- 最大间隔聚类问题可以用 Kruskal 算法解决
- Kruskal 算法
 1. 初始化 n 个连通分支，**对应 n 个聚类**
 2. 从小到大枚举所有边 e
若加入 e 不会出现环，则加入 e ，等价于**合并两个连通分支**
 3. 如果只剩下一个连通分支，停止（得到最小生成树）
- 如果修改 Kruskal 的终止条件：如果剩下 k 个连通分支，停止。则可以得到符合题意的 k 个非空子图

由于 Kruskal 算法每次加入新边时都是考虑权值最小的边，因此，当得到 k 个连通分支时，**还未加入的 $k-1$ 条边**中其实就是最小生成树中距离最大的 $k-1$ 条边，因此，当去掉这最长的 $k-1$ 条边时得到的这 k 个聚类的间隔也是最大的

- 这里的“还未加入的 $k-1$ 条边”是指还未加入最小生成树的 $k-1$ 条边
合并两个连通分支需要加入一条边，因此合并剩下的 k 个连通分支需要 $k-1$ 条边
- 示例



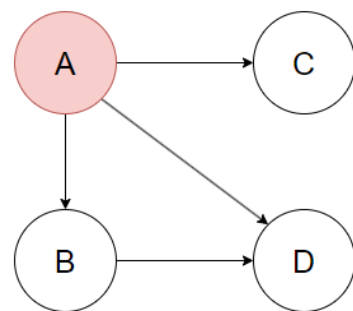
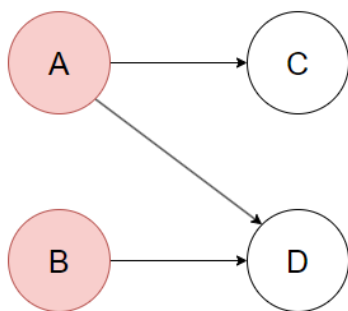
例题：二部图

6、无向图 $G=(V,E)$ 是二部图是指该图的点集 V 能划分成非空的两部分 V_1 和 V_2 使得图 G 中任何边的两个端点分别属于 V_1 和 V_2 。考虑用图的深度优先遍历方法设计一个算法判别无向图 $G=(V,E)$ 是否是二部图。要求：

- (1) 用文字简要描述算法的基本思路（不用写代码）。
- (2) 给出算法正确性的证明。
- (3) 如果要求利用宽度优先遍历图的方法设计算法判别图 $G=(V,E)$ 是否是二部图呢？

(1) 文字描述算法思路

- 通过从某点开始对图进行 DFS 并依次给图中相邻节点分别着两种颜色（节点的“访问”即为节点的“着色”）
- 同一个节点不会重复着色
- 在此过程中如果存在有两个端点同色的回边，则该图不是二部图，否则该图是二部图
 - **回边**：在 DFS 过程中，发现的一个边，该边连接到当前节点的祖先节点
- 示例



- 左图
假设从 A 开始 DFS

$\rightarrow C$
 $A \rightarrow D$
 $B \rightarrow D$

着色过程中未出现回边同色，**左图是二部图**，其中 $V_1 = \{A, B\}, V_2 = \{C, D\}$

- 右图

假设从 A 开始 DFS

$\rightarrow C$
 $A \rightarrow D$
 $A \rightarrow B \rightarrow D$

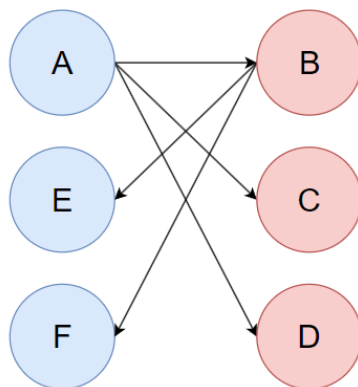
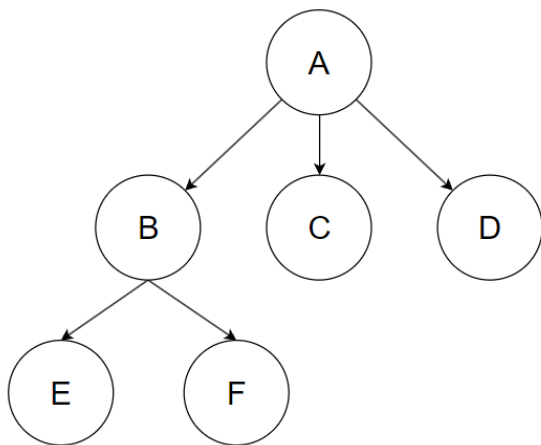
在 $A \rightarrow B \rightarrow D$ 中，遍历到 B 时，D 已经着色，且和 B 同色，而 $\langle B, D \rangle$ 是一条回边，因此**右图不是二部图**

(2) 证明算法正确性

- 算法：图着色后，若不存在端点同色的回边，则是二部图，否则不是
- 证明
 - 必要性：如果图 G 是二部图，则所有边都跨于 V_1, V_2 之间，且算法中给 V_1 中所有点设置的是同一种颜色，给 V_2 的是另一种颜色。因此不存在两个端点同色的回边
 - 充分性：将不同颜色的节点分为 V_1, V_2 ，因为不存在端点同色的回边，所以任意边都跨于 V_1, V_2 之间，因此 G 是二部图

(3) BFS

- 算法：用 BFS 遍历图 G，得到 BFS 生成树，若生成树中同层节点之间没有边，则是二部图，否则不是
 - BFS 生成树示例



生成树同层节点无边，是二部图，可以将点分为 $V_1 = \{A, E, F\}, V_2 = \{B, C, D\}$

- 证明：
 - 必要性：若 G 是二部图，将奇数和偶数层的节点分别分配到 V_1, V_2 ，因为所有边都跨于 V_1, V_2 之间，所以 BFS 生成树中只有跨层的边
 - 充分性：将奇数和偶数层的节点分别分配到 V_1, V_2 ，由于同层之间没有边，因此不存在一条边，该边的端点都属于 V_1 或 V_2 ，因此 G 是二部图

例题：背包问题

7、使用动态规划法求解下列 0-1 背包问题实例：物品个数 $n=4$ ，重量 $W=\{2,3,4,5\}$ ，利润 $P=\{3,4,5,7\}$ ，背包承重量 $C=9$ 。

- (1) 给出递推计算的公式。
- (2) 用表格或图展示出主要计算过程，并指出问题的解。

(1) 递推公式

- 状态定义： $dp[i][j]$ 表示考虑 $1 \sim i$ 个物品，重量不超过 j 的最大利润
- 状态计算： $dp[i][j] = \max(dp[i-1][j], dp[i-1][j-w[i]] + p[i])$

(2) 计算过程

物品/重量	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7	7	7
3	0	0	3	4	5	7	8	8	9	12
4	0	0	3	4	5	7	8	10	11	12

- 考虑 $1 \sim 4$ 个物品，重量不超过 9 的最大利润为 $dp[4][9] = 12$
具体的选法组合有两种： $[1, 2, 3], [3, 4]$

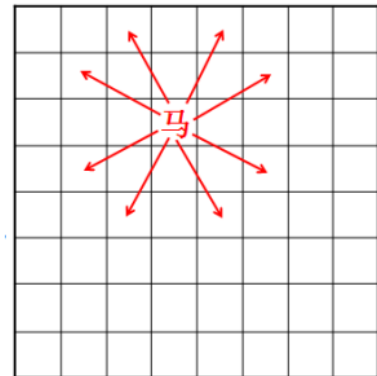
例题：马周游问题

8、【马周游问题】在 8×8 棋盘上马按照跳日字的规则能从一格跳到另一格。任给马在棋盘上的起始位置，请问是否存在一种马能跳到棋盘中每个格子恰好一次且最后回到起始位置的周游棋步？如果存在，请找出一种这样的周游棋步。

(1) 用图论知识说明：若马从棋盘正中心位置出发能有周游棋步，则马从棋盘的任意位置出发都有周游棋步。

(2) 考虑用回溯法求解马周游问题。请给出解向量形式和搜索树类型，描述剪枝操作。

(3) 可采取哪些措施来提高回溯算法的求解效率？



图论知识

- 哈密顿路径：图中经过每个顶点恰好一次的路径
- 哈密顿圈：在一条哈密顿路的基础上，再有一条边将其首尾连接，所构成的圈
- 哈密顿图：有哈密顿圈的图
- 参考文献
 - [哈密顿图 Wiki](#)

回溯法

- 解向量: n 元组 (x_1, x_2, \dots, x_n)
- 隐式约束: 问题给定的约束条件
- 显式约束: 解向量分量的 x_i 的取值范围
- 解空间: 满足显式约束的一组解向量, 回溯法的解空间可以组织成一棵树

(1) 证明若存在从中心出发的周游棋步, 则从任意位置出发都有周游棋步

- 将棋盘的格子看作一个节点, 马能够跳到的两个格子是一条无向边
- 存在从中心出发的周游棋步等价于存在**哈密顿圈**
- 由图论知识可知, 若有一个哈密顿圈, 则移除其任一条边, 皆可得到一条哈密顿路, 将哈密顿路的起点和终点进行连接可得到哈密顿圈
- 所以若马从棋盘正中心位置出发能有周游棋步, 则马从棋盘的任意位置出发都有周游棋步

(2) 回溯法解决马周游问题

- 解向量: (x_1, x_2, \dots, x_n) , 其中 $1 \leq x_i \leq 8 (1 \leq i \leq n)$
- 搜索树: 64 层高的满 8 叉树
- 剪枝操作
 1. 马不能跳到已访问的格子
 2. 保证马跳跃后的位置在棋盘范围内

(3) 可采取哪些措施来提高回溯算法的求解效率

- 先求出马从棋盘中心出发的一条周游棋步, 并保存起来。然后根据这条周游棋步通过位置的变换容易得到从任何初始位置出发的周游棋步
- 在回溯法的搜索过程中, 优先选择跳跃到出度最小 (可供跳跃的选择最少) 的位置

例题：设备购买与维护

9、工厂因为开展新业务打算明年年初新购置某台设备,并要做好后续 5 年该设备的更新计划,决定每年是重新购置该设备还是继续维护现有设备.已经预测今后 5 年该设备的购置费和维护费分别如下表 1 和表 2 所示(购置价格和维护费用的单位: 万元).请设置一个该设备的 5 年更新方案,使得在此期间该设备的购置费和维护费的总和最小.

表 1 每年的设备购置费用(单位: 万元)

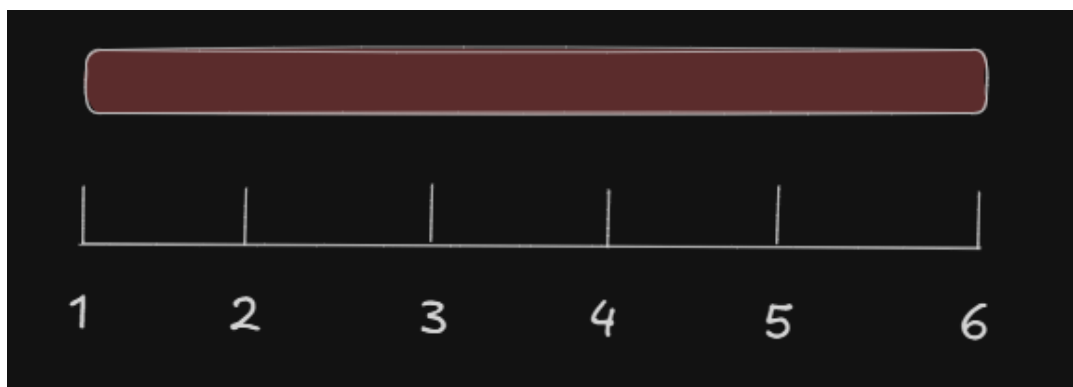
第 1 年	第 2 年	第 3 年	第 4 年	第 5 年
22	22	24	24	26

表 2 设备维护费用(单位: 万元)

0-1 年	1-2 年	2-3 年	3-4 年	4-5 年
10	12	16	22	36

分析

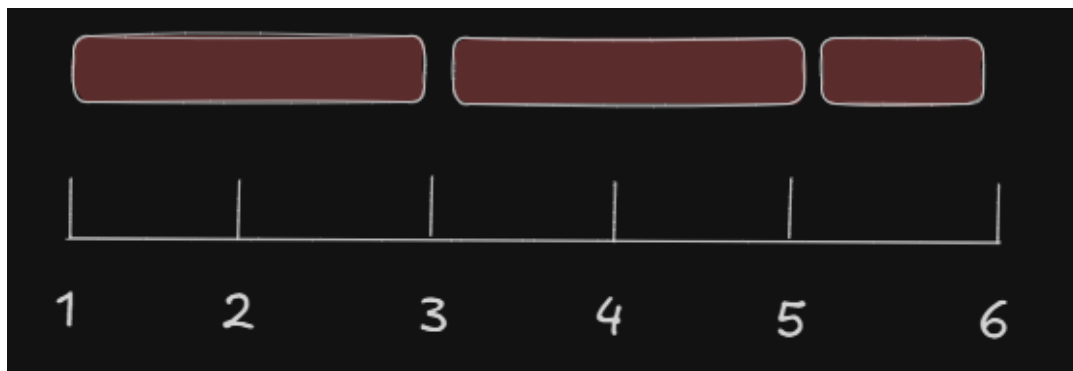
- 工厂每年只用 1 台设备, 如果买了新设备, 则旧设备不再维护
- 从第一年到第六年都要有设备可用
- 举例
 - 第 1 年购买设备, 连续用 5 年



红色区域表示维护时间

总费用：第 1 年购买费 + 5 年维护费 = $22 + (10 + 12 + 16 + 22 + 36) = 118$

- 隔年购买设备



红色区域表示维护时间，间断处表示购买新设备，不再维护旧设备

总费用：第 1,3,5 年购买，分别维护 2,2,1 年 = $(22 + 24 + 26) + (22 + 22 + 10) = 126$

不管怎样，第一年都必须买设备

- 将问题转化为求最短路
 - 节点 x ：购买第 x 年的设备
 - 边 $\langle x, y \rangle$
 - 购买第 x 年的设备，用到第 y 年
 - 权重即总费用
 - 寻找总费用最少的设备更新方案问题转为求从 v_1 到 v_6 的最短路径问题
 - 构建有向图

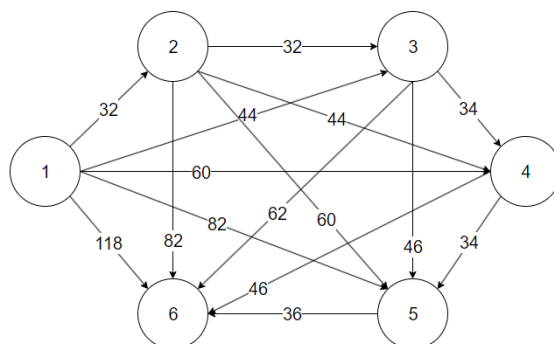


表 1 每年的设备购置费用(单位: 万元)

第 1 年	第 2 年	第 3 年	第 4 年	第 5 年
22	22	24	24	26

表 2 设备维护费用(单位: 万元)

0-1 年	1-2 年	2-3 年	3-4 年	4-5 年
10	12	16	22	36

$\langle x, y \rangle$ = 第 x 年的购置费 + 0~ $y-x$ 年的维护费用总和

- 使用 Dijkstra 算法求最短路，最低总费用为：106（万元），共有两个方案：

$$1 \rightarrow v_3 \rightarrow v_6$$

$$v_1 \rightarrow v_4 \rightarrow v_6$$

代码

- 数组模拟邻接表 + 前缀和

```
1  #include <iostream>
2  #include <cstring>
3  #include <map>
4  #include <vector>
5  using namespace std;
6
7  const int N = 1e3;
8  int h[N], e[N], ne[N], w[N], idx;
9
10 void add(int a, int b, int c) {
11     e[idx] = b;
12     ne[idx] = h[a];
13     w[idx] = c;
14     h[a] = idx++;
15 }
16
17 const int n = 5;
18 const int m = n + 1;
19 int buy[] = {0, 22, 22, 24, 24, 26};
20 int fix[] = {0, 10, 12, 16, 22, 36};
21 int fix_sum[m]; // 前缀和
22
23 void build_graph() {
24     memset(h, -1, sizeof h);
25     // 构建前缀和数组
26     for(int i=1; i<=n; ++i) {
27         fix_sum[i] = fix[i] + fix_sum[i-1];
28     }
29     // 建图
30     for(int x=1; x<=n; ++x) {
31         for(int y=x+1; y<=m; ++y) {
32             add(x, y, buy[x] + fix_sum[y-x]);
33         }
34     }
35 }
36
37 void show_all_paths(int cur, map<int, vector<int>>& path, vector<int>& temp)
38 {
39     temp.push_back(cur);
40     if(path[cur].empty()) {
41         for(int i=temp.size()-1; i>=0; --i) {
42             if(i != temp.size()-1) cout << " -> ";
43             cout << temp[i];
44         }
45         cout << endl;
46     } else {
47         for(int from : path[cur]) {
48             show_all_paths(from, path, temp);
49             temp.pop_back();
50         }
51     }
52 }
```

```

53
54
55 int dist[N];
56 bool visit[N];
57 void dijkstra(int start) {
58     const int inf = 0x3f3f3f3f;
59     memset(dist, inf, sizeof dist);
60     dist[start] = 0;
61     map<int, vector<int>> path;
62     while(true) {
63         int x = -1, cost = inf;
64         for(int i=1; i<=m; ++i) {
65             if(!visit[i] && dist[i] < cost) {
66                 cost = dist[i];
67                 x = i;
68             }
69         }
70         if(x == -1) break;
71         visit[x] = true;
72         for(int i=h[x]; i!=-1; i=ne[i]) {
73             int j = e[i];
74             int d = dist[x] + w[i];
75
76             // 只存最短路径或同样短的路径
77             if(d < dist[j]) {
78                 dist[j] = d;
79                 path[j].clear();
80                 path[j].push_back(x);
81             } else if(d == dist[j]) {
82                 path[j].push_back(x);
83             }
84         }
85     }
86
87     vector<int> temp;
88     show_all_paths(m, path, temp);
89
90     cout << dist[m] << endl;
91 }
92
93 int main() {
94     build_graph();
95     dijkstra(1);
96     return 0;
97 }

```

- 输出

```

1 1 -> 3 -> 6
2 1 -> 4 -> 6
3 106

```

三、NP 完全性理论基本概念

如何理解求解问题的具体算法的复杂度、问题的复杂度

- 算法的复杂度：基本运算执行次数，包括时间复杂度和空间复杂度
- 问题的复杂度：求解这个问题的最优算法的复杂度

什么是归约 (reduce)

- 若有解决 B 问题的算法，而 A 问题可以转化为 B 问题，则称 A 可以（多项式时间）归约为 B
- 符号： $A \leq B$ ，可以在 \leq 符号下标指定归约类型（m：映射缩小，p：多项式缩减）
问题 A 的难度不超过 B
- 对于问题 A 和 B，若有一个函数 f，对于 A 的任意实例 x，都有 $A(x) = B(f(x))$
- A 归约到 B 主要做两件事：
 1. 将问题 A 的输入转化为问题 B 的输入
 2. 将问题 B 的输出转化为问题 A 的输出
- 举例：乘法可以归约到平方（假设只能加、减、平方、除以2）

$$a \times b = \frac{(a+b)^2 - a^2 - b^2}{2}$$

当然也可以从另一个方向归约：平方归约到乘法

$$a^2 = a \times a$$

这两个问题难度基本相等，称为图灵归约

归约在建立问题的复杂度方面有何作用

- 算法设计角度
若有 $A \leq B$ ，则问题 A 可以转化为问题 B 求解
- 计算复杂性角度
若有 $A \leq B$ ，则问题 B 的难度不低于问题 A 的难度
因此归约可以比较两个问题的难度

说明语言类 P、NP、NP-complete、NP-hard 的含义

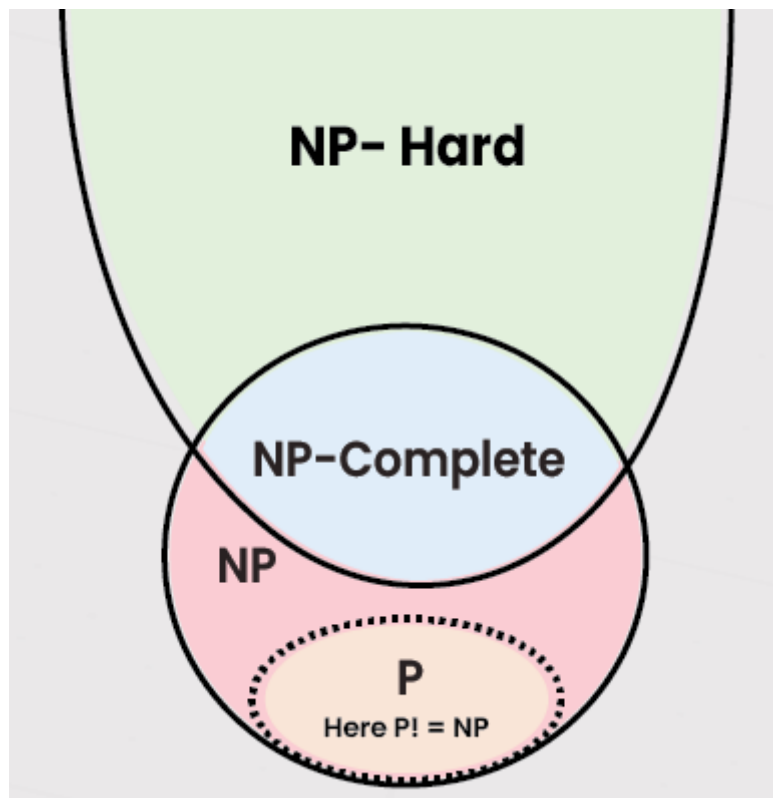
- 多项式

$$0 + x_1n + x_2n^2 + x_3n^3 + \dots + x_kn^k$$

- P (Polynomial Time)
 - 能够在多项式时间内解决的问题
- NP (Nondeterministic Polynomial Time)
 - 不确定是否存在多项式时间的求解算法
 - 可以在多项式时间内验证其解是否正确
 - 举例：数独、求图的哈密顿回路
 - 能在多项式时间内解决的问题一定能在多项式时间内验证，因此 P 问题一定是 NP 问题
- NP-hard, NPH
 - 任意 NP 问题可以在多项式时间归约成该问题
 - 举例：旅行商求最短回路
可以在多项式时间内求出花费，但无法确定是否是最低花费，因此 NP-hard 问题不一定是 NP 问题
- NP-complete, NPC

- 所既是 NP 问题，又是 NP 难问题的问题
- 举例：旅行商在限制花费时是否可行
任意给出一个行程安排，可以验证花费是否小于指定花费

假定 $P \neq NP$ ，画图表示其包含关系



SAT 问题

- 布尔可满足性问题 (Boolean satisfiability problem; SAT)
- 对于一个 bool 表达式，是否存在某个输入使得整个表达式为 True
 - 常见符号
 - 合取： \wedge ，等价于 AND
 - 析取： \vee ，等价于 OR
 - 否定： \neg 或 \neg ，等价于 NOT
 - 举例： $(x_1 \wedge \overline{x_2}) \vee x_3$ ，当 $x_1 = \text{True}, x_2 = \text{False}, x_3$ 任意时或 $x_3 = \text{True}, x_1, x_2$ 任意时，表达式为 True
- 这是一个 NPC 问题

证明一个问题是 NP-hard 有哪些方法

1. 若某问题的难度不低于任何一个 NP 问题，则该问题是 NP-hard 问题
2. 将已知的 NP-hard 问题归约到该问题

如何处理 NP 难问题

- 无法求最优解，那就求近似最优解
 1. 精确解
 - 分治、动态规划、分支限界、回溯等
 2. 近似解

随机、松弛等

3. 启发式

贪心、模拟退火等

指出下面关于 $P \neq NP$ 错误证明的错处，并说明你认为它是错误的理由

- 证明：考虑 SAT 的一个算法：“在输入 ϕ 上，尝试变量的所有可能的赋值，若有满足 ϕ 的就接受”。该算法显然需要指数时间。所以 SAT 有指数时间复杂度，因此 SAT 不属于 P。因为 SAT 属于 NP，所以，P 不等于 NP。
- 错在混淆 SAT 问题的时间复杂度和其一个具体算法时间复杂度

n 点独立集问题

- 独立集：在无向图 G 中的一个点集 S，S 中任意两个顶点之间没有边相连
- n 点独立集问题：判断图中是否存在大小为 n 的独立集，该问题是 NP 难度的

n 皇后问题如何归约为在一个 n^2 点无向图中找一个 n 点独立集的问题

- 设问题 A：n 皇后问题，问题 B：在一个 n^2 点无向图中找一个 n 点独立集的问题
- n^2 个点对应 $n \times n$ 棋盘中的每个位置，对所有横向、竖向、斜向的任意两个点进行连边
- 若有一个大小为 n 的独立集 S，则该独立集的任意两点 $u, v \in S$ 不再同一行、同一列、同一个斜向，满足 n 皇后解定义
- 则 A 可以归约为 B

既然图的独立集问题是 NP 难度的，能否由该归约得出 n 皇后问题是 NP 难度的

- 不能，归约方向不对： $A \leq B$ 是指 A 不难于 B，B 是 NP 难度不代表 A 也是
- 并且皇后问题存在多项式时间解，属于 P 问题

四、NP 完全性理论的基本方法（归约技术）

NP-hard 证明

为证明 A 是 NP-hard，需要将一个 NP-hard 难度的问题 B 归约到 A，即 $B \leq A$

NP-Complete 证明

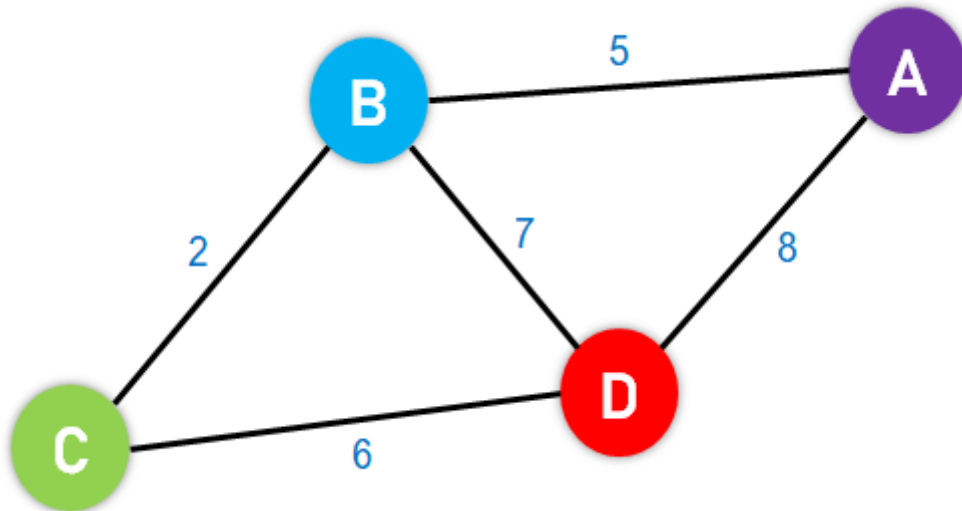
- 前提：已知 A 是 NPC 问题
- 方法
 1. 若已知 B 是 NP 问题
证明 $A \leq B$ ，则 B 是 NP-hard 问题，又由前提可以推出 B 是 NPC 问题
 2. 若已知 A 是 B 的特例（有 $A \leq_p B$ ）
则 B 是 NPC 问题

例题：最短路

1、图 $G = (V, E, W)$ 中给定起点和终点 $s, t \in V$ 和中间节点集 $D \subset V$ ，要求找出从 s 到 t 且中间须经过 D 中每节点最少 1 次的最短路径。如何求解？

1. 使用最短路径算法求出 $V' = \{s, t\} \cup D$ 中任意两点的最短路径 $dist[i][j]$ (i 到 j 的最短路径)
2. 以 V' 为点集, $\langle i, j \rangle, i, j \in V'$ 为边集, $dist[i][j]$ 为边权, 构建一个新图 G'
3. 原问题归结为在 G' 中求一条以 s 为起点 t 为终点经过图中每个点至少一次的最短路径

• 示例

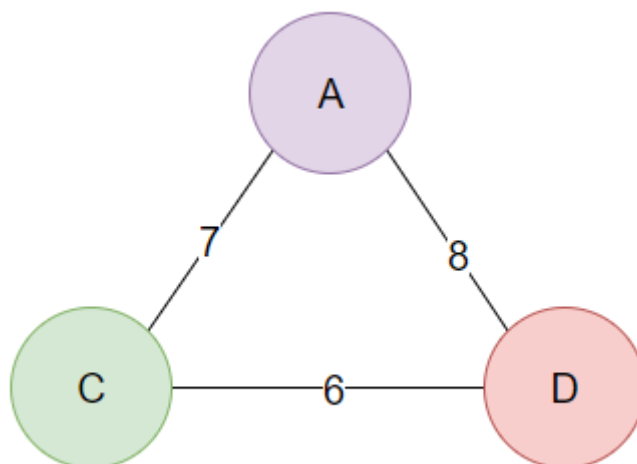


假设 $s = A, t = C, D = \{D\}$, 一眼看出最短路径是 $A \rightarrow D \rightarrow C$, 长度为 $6 + 8 = 14$

1. $V' = \{A, C, D\}$, 求最短路

起点	终点	距离
A	C	7
A	D	8
C	D	6

2. 构建新图



由于举例的是无向图, 因此 $dist[i][j] = dist[j][i]$

3. 从 A 出发, 到 C 且经过所有点的最短路径为 $A \rightarrow D \rightarrow C$, 长度为 $8 + 6 = 14$

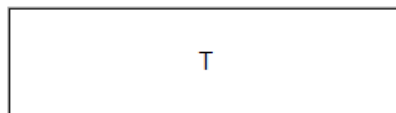
例题：Partition 和 SubsetSum 等价

2、证明: $\text{Partition} \leq \text{SubsetSum}$ 且 $\text{SubsetSum} \leq \text{Partition}$, 即 $\text{Partition} \equiv \text{SubsetSum}$ 。

- **Partition** 问题: 给定一组整数, 是否能将这组整数划分为两个子集, 使得两个子集的元素和相等?
- **SubsetSum** 问题: 给定一个整数集合和一个目标值, 是否存在一个子集, 使得该子集的元素之和等于目标值?
- 证明

◦ $\text{SubsetSum} \leq \text{Partition}$

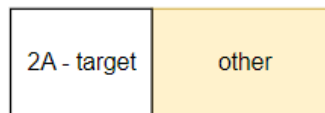
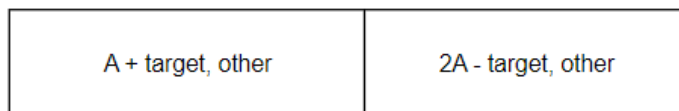
1. 将 SubsetSum 的输入 (一个整数集合和一个目标值) 转化为 Partition 的输入 (一组整数)
 - 给出 SubsetSum 的一个输入实例: 整数集合 $T = (t_1, t_2, \dots, t_n)$, 目标值 $target$
 - 设 $A = \text{sum}(T)$, 一个新集合 $W = (T, 2A - target, A + target)$, 令 W 为 Partition 的输入
2. 将 Partition 的输出转化为 SubsetSum 的输出
 - W 会被划分成两个和相等的子集 W_1, W_2 , 且 $\text{sum}(W) = 4A$, 因此 $\text{sum}(W_1) = \text{sum}(W_2) = 2A$, 由于 $(2A - target) + (A + target) = 3A$, 故这两个元素不能在同一个子集中
 - $t_{n+1} = 2A - target$ 所在的子集的和是 $2A$, 因此除去 t_{n+1} 后的其他元素构成的集合之和为 $target$, 符合 SubsetSum 的定义



target



sum = 4A



sum(other) = target

◦ $\text{Partition} \leq \text{SubsetSum}$

1. 将 Partition 的输入转化为 SubsetSum 的输入
 - 给出 Partition 的一个输入实例: 整数集合 $T = (t_1, t_2, \dots, t_n)$
 - 设 $\text{sum}(T) = A$, 令集合为 T , 目标值为 $A/2$ 为 SubsetSum 的输入
2. 将 SubsetSum 的输出转化为 Partition 的输出
 - 得到一个子集 W , 其中 $\text{sum}(W) = A/2$
 - T 划分出的两个子集和相等, 那么每个子集的和都是 $A/2$, 故 Partition 要的两个子集是 W 和 $W' (i \in W', i \notin T)$

◦ 故 $\text{Partition} \equiv \text{SubsetSum}$

- [参考资料](#)

例题：证明负载均衡问题是NP完全的

3、试根据Partition、SubsetSum的NP完全性，证明负载均衡问题是NP完全的。

- 负载均衡问题（Load Balancing Problem）：给定一组任务和若干台机器，判断是否能够将任务分配给机器，使得所有机器的负载（即任务总和）相等
- 当机器数=2 时，负载均衡问题转化为 Partition 问题，即 $Partition \leq LBP$
而 Partition 问题是 NPC 问题，所以 LBP 是 NPC 问题

例题：点覆盖问题 \leq Roman-Subset 问题

4、证明：点覆盖问题 \leq Roman-Subset 问题

【点覆盖问题】给定无向图 $G=\langle V, E \rangle$ ，求最小规模点子集 $S \subseteq V$ 满足 S 能覆盖 E 中所有边。

【Roman-Subset 问题】给定一个有向图 $G=\langle V, E \rangle$ ，求最小规模点子集 $S \subseteq V$ 满足 G 中任何回路上都至少有一点在 S 中。

- 点覆盖 \leq Roman-Subset
 1. 将点覆盖的输入转化为 Roman-Subset 的输入
 - 给定点覆盖的一个输入实例：无向图 $G = \langle V, E \rangle$
 - 设一个有向图 $G' = \langle V, E' \rangle$ ，其中 E' 为 E 中的边 $\langle u, v \rangle$ 等价的 $\langle u, v \rangle, \langle v, u \rangle$ ，令 G' 为 Roman-Subset 的输入
 2. 将 Roman-Subset 的输出转化为点覆盖的输出
 - 任意 $\langle u, v \rangle \in E'$ 中，最多只有一点在 S 中，且 $|E'| > |E|$ ，故 S 也是 G 的最小点覆盖子集

例题：哈密顿圈问题 \equiv 哈密顿路问题

5、证明：（有向图）Hamilton圈问题 \leq Hamilton路问题，且 Hamilton路问题 \leq Hamilton圈问题，即 Hamilton路问题 \equiv Hamilton圈问题。

- A 哈密顿路问题（一笔画问题）：图中是否存在经过每个顶点恰好一次的路径
- B 哈密顿圈问题：图中是否存在起点和终点相同，且经过每个（其他）顶点恰好一次的路径
- $A \leq B$
 1. 将 A 的输入转化为 B 的输入
 - A 的输入实例： $G = \langle V, E \rangle$
 - 令 G 为 B 的输入
 2. 将 B 的输出转化为 A 的输出
 - 存在哈密顿圈 $v_1, v_2, \dots, v_n, v_1$ ，则 v_1, \dots, v_n 为哈密顿路
- $B \leq A$
 1. 将 B 的输入转化为 A 的输入
 - B 的输入实例： $G = \langle V, E \rangle$
 - 对于 G 中的任意点 v ，用 v_1, v_2 替换，所有指向 v 的边都指向 v_1 ，所有指出 v 的边都由 v_2 指出，构成的新图为 G'
 - 令 G' 为 A 的输入
 2. 将 A 的输出转化为 B 的输出
 - 存在哈密顿路 v_i, \dots, v_j
 - 由于 v_2 无入边， v_1 无出边，故 $v_i = v_2, v_j = v_1$ ，则哈密顿路为 $v_2, v_k, \dots, v_l, v_1$
 - 则在 G 中存在一条路径 $v, v_k, \dots, v_l, \dots, v$ ，该路径为哈密顿圈

- 故 $A \equiv B$