从米开朗基罗到算法的三种设计策略

原创 蔡秀定 LOA算法学习笔记 2021-02-10 16:30

1 从米开朗基罗开始

米开朗基罗是意大利著名的雕塑家，与拉斐尔和达芬奇并称为文艺复兴后三杰，是文艺复兴时期雕塑艺术最高峰的代表。

这样一个雕塑大师，米开朗基罗是如何雕塑的？

Michelangelos's art: A major part of his effort involved looking for interesting pieces of stone in the quarry and staring at them for long hours to determine the form they naturally wanted to take. The chisel work exposed, in a minimal manner, this form.

原来，米开朗基罗大部分时间花在了采石场找一块感兴趣的石头，然后花数个小时观察它的天然结构。一旦看清楚了这块石头所内蕴的天然结构之后，斧凿之功就是简单的事情了。

这与算法的设计是类似的。一个待解决的问题就好比米开朗基罗在采石场找到的石头，我们应该花大部分时间来研究其性质和相关的结构，一旦这些清楚后，算法的设计在盏茶时间。换言之，算法设计的关键在于深入观察待解决问题的特性；==**我们对问题特性认识得越多，可资使用的算法技术就越多，设计出的算法也就可能更好。**

2 算法设计流程

上面提到了算法设计一个重要的步骤：观察，但怎么观察、观察什么、什么时候观察等问题都没有讨论，在这一节，回顾卜老师在第一节课讲的算法设计流程。

首先，对于一个实际问题（practical problem），首先尝试将其抽象、形式化为一个数学问题（math problem）。

在这一步的基础上，观察。

观察，观察什么呢？一是观察问题，二是观察问题的解。

观察问题又可分为三小点来观察

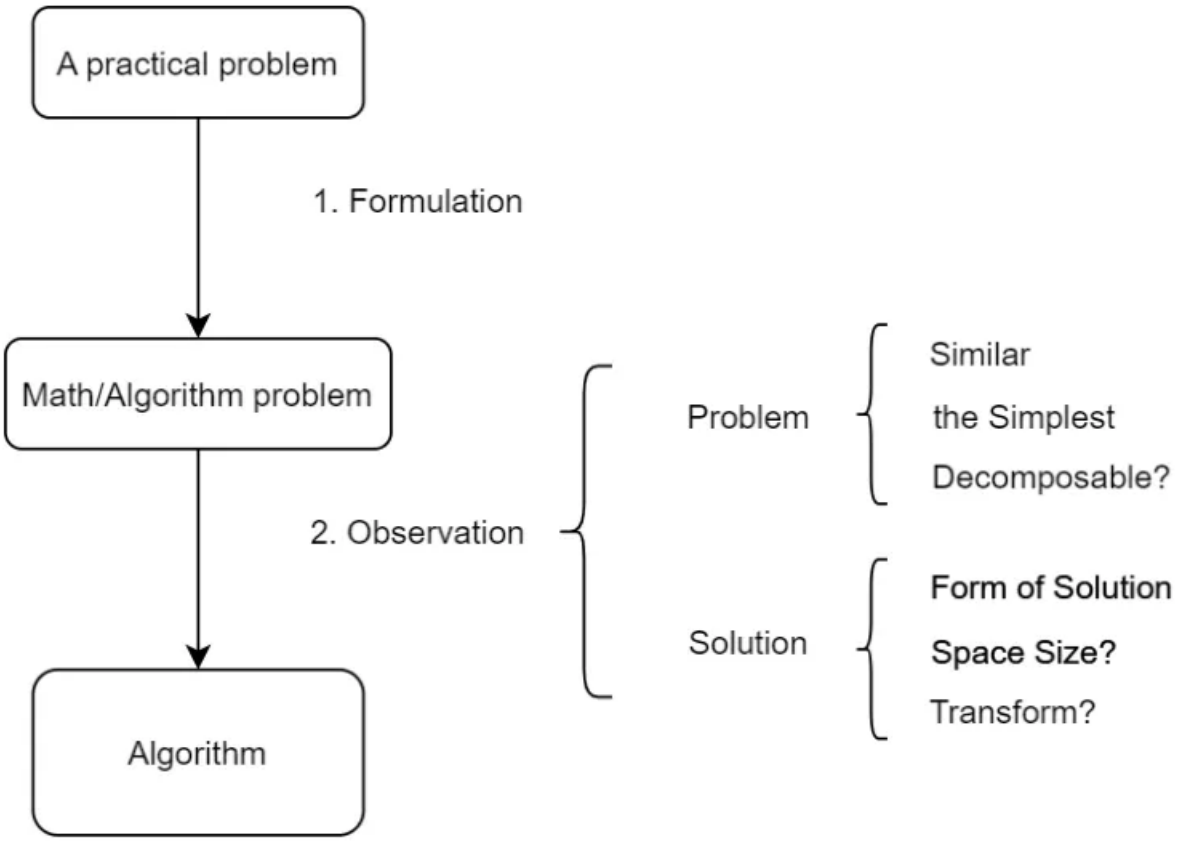
1. Similar：有没有类似的问题？可不可以转换为类似的问题？如果有类似的问题，但不能转换，是什么阻碍了转换？
2. Simplest：这个问题的最简单的 case 是什么？能不能从简单的 cases 找到算法设计的启发？
3. Decomposable：复杂问题能不能分解？分解后是否会更简单？

观察问题的解也可分为三小点来观察

1. Form of Solution：问题的可行解的形式是什么？
2. Space Size：可行解的总数有多少？
3. Transform：可行解能否一步一步地逐步构建出来？我们能否对一个可行解施加小幅扰动，将之变换成另一个可行解？

最终，在观察的基础上，我们设计出相应的算法。

总的流程图如下图所示。



3 三种算法设计策略

在上一节回顾了算法的设计流程，并着重介绍了“对问题的观察”。在这一节，回顾如何依据对问题内在结构的观察，采用相应的策略设计算法。

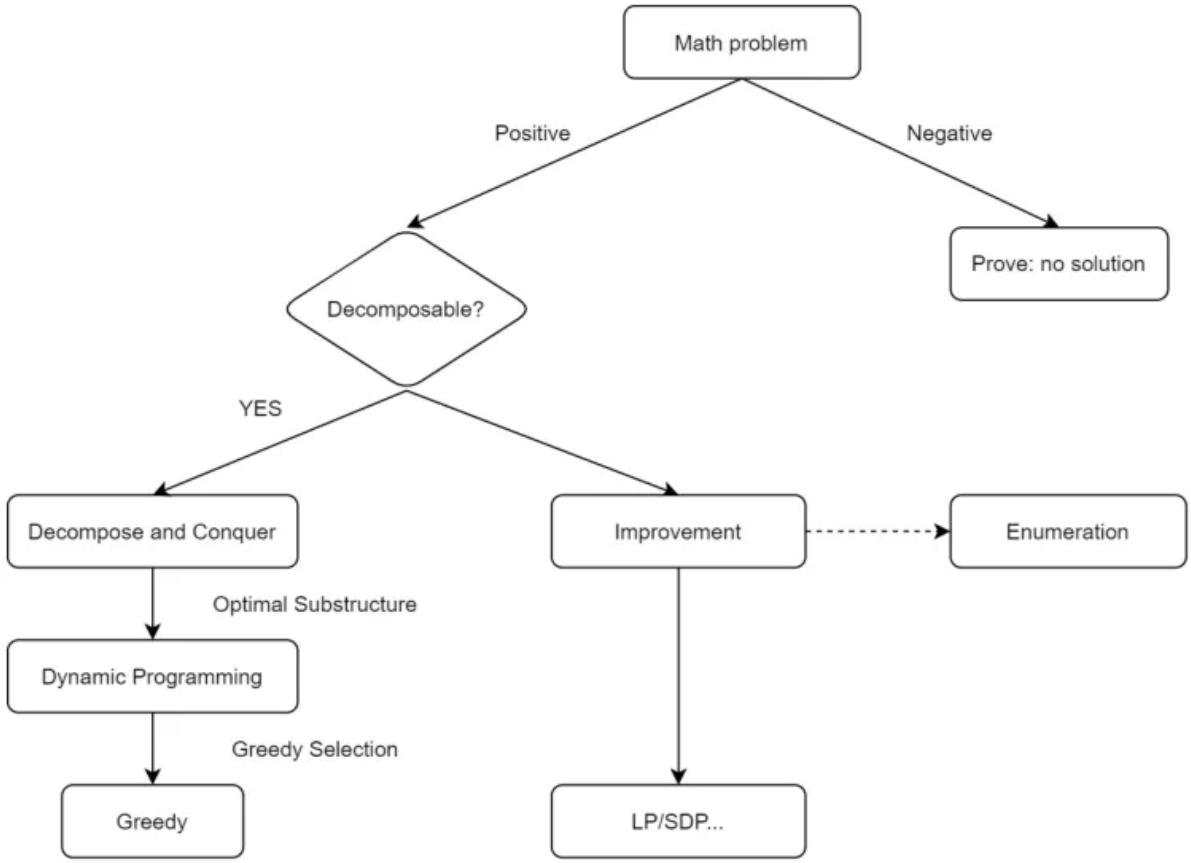
对于一个 math problem，我们可以用一周的时间尝试正面求解，如果求解不出，会不会**不存在**解呢？我们可以再用一周的时间证明这件事，依次反复。

卜老师在这里传达了一个很重要的科研思想：科研是到无人之境探索未知事物，对于一个问题，没有人知道它有没有解，这也正是科研和我们平时做练习题的一个重要区别，我们必须意识到这一点，这很重要。

求解一个 math problem 通常有三种策略

- reduction-based strategy：基于分而治之的策略
- improvement-based strategy：基于逐步改进的策略
- enumeration-based strategy：基于（智能）枚举的策略

下面我们依次介绍这三种策略的哲学。



首先，我们尝试解决这个问题的最简单的形式，如果最简单的形式都没有思路，也不用做了。
做完这一步后，我们接下来可以思考这个问题能不能分解为子问题。
第一种策略：如果可分解，这意味着我们可以使用归纳（Reduction）的方法求解。
归纳方法中比较典型的是分而治之（divide and conquer，DC）的方法。
继续观察，如果不仅具有可分解的 property，还具备最优子结构（optimal substructure）的性质，那么我们可以使用动态规划（dynamic programming，DP）的方法。
注：最优子结构是指我们能用子问题的最优解组合成原始问题的最优解。（需要回溯）
再进一步地观察问题结构：如果待求解的问题不仅具有最优子结构性质，还具有 greedy selection 性质，那我们可以尝试设计贪心算法求解问题。
所谓贪心选择性质，是指局部最优决策 (Locally optimal decision)，可以直观地理解为做决策时的"短视"策略，即：问题的解可以一步一步地逐渐构造而成；在构造的每一步，无需考虑尚未求解的子问题，只依据已经构造出的部分解即可做出最优或者近乎最优的选择。
在这一点上，贪心算法与动态规划算法显著不同：动态规划算法中需要考虑子问题的解、通过回溯才能确定出最优解。
第二种策略：如果问题不能分解，或可分解但不愿意分解（如网络流问题迄今为止没有找到分解后可高效计算的算法），我们此时只能采用基于改进的策略（improvement-based strategy）。
这种策略的思想是，在完备的解空间中，首先初始化得到某个解，然后基于某种策略（这种策略是基于解与解之间的关系设计的），不断求出新的解，如果新解优于当前解，则用新解替代当前解，以此不断逼近解空间中的最优解。
这里面典型的算法是：

- 线性规划（LP）、二次规划（QP）、半正定规划（SDP）、
- 网络流（NF）、
- 局部搜索（LS）、
- 蒙特卡洛（MC）、模拟退火算法（SA）。

注：SA 是 MC 的随机化版本。
第三种策略：如果我们观察到 form of solution 可以写成 $X = [x_1, x_2, \cdots, x_n]^T, x \in \{0, 1\}$ 的形式，这意味着我们可以使用基于枚举的策略（enumeration-based strategy）。
如果每个解由 n 个元素构成，每个元素只能为 0 或 1，则解空间大小为 2^n 。如果是完全枚举的话，这种策略是非常耗时间的。因此有必要在枚举的过程中进行 "剪枝"。

枚举类的典型算法有回溯法（Backtracking）、分支限界算法（Branch and bound）等。值得指出的是，贪心算法可以看做枚举策略的一个特例：贪心算法在构造解的每一步都依据贪心规则作出选择，最终获得解实质上是枚举树中的一条路径，因此可以看做是最极端的一种剪枝。

以上就是求解 math problem 的三种策略，卜老师：也只有这三种策略。

针对"难"的问题的算法设计策略：对于有些问题，我们不可求解，或者我们可以证明其为一个 Hard 问题。

对于 Hard 问题，不强求解，我们可以适当放松要求（trade-off）：

- optimal → approximation：不再要求最优解
- deterministic → randomization：不再要求算法的每一步是确定性的，可以加入一些随机步骤，当然这会造成一些问题，如求解速度时快时慢等
- all cases → practical case：不再要求算法适用于所有情况

但不管怎么说，Hard 问题的求解依然只有上述的三种策略。

4 例子：旅行商问题

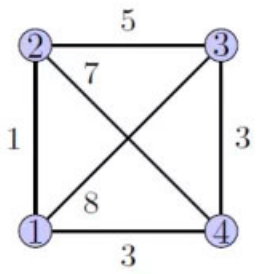
在上一节，介绍了算法设计的三种策略。在这一节，将通过著名的旅行商问题（traveling salesman problem，TSP），依次介绍这三种策略的应用。

旅行商问题说的是：给定 n 个城市，从任意一个城市出发，遍历完其余的城市（每个城市只经过一次），最后回到出发城市，要求行程费用最小。其中直接毗邻的两个城市之间存在行程费用。

问题的形式化定义如下：

Travelling Salesman Problem

INPUT: n cities $V = \{1, 2, ..., n\}$, and a distance matrix D, where d_{ij} $(1 \leq i, j \leq n)$ denotes the distance between city i and j.
OUTPUT: the shortest tour that visits each city exactly once and returns to the origin city.

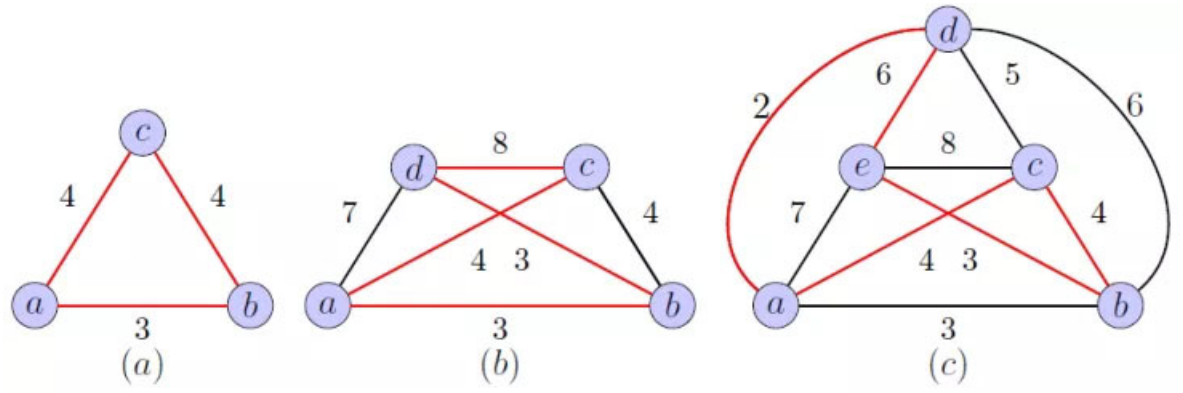


- Tour 1: $1 \to 2 \to 3 \to 4 \to 1$ (distance: 12)
- Tour 2: $1 \to 2 \to 4 \to 3 \to 1$ (distance: 19)
- Tour 3: $1 \to 3 \to 2 \to 4 \to 1$ (distance: 23)
- Tour 4: $1 \to 3 \to 4 \to 2 \to 1$ (distance: 19)
- Tour 5: $1 \to 4 \to 2 \to 3 \to 1$ (distance: 23)
- Tour 6: $1 \to 4 \to 3 \to 2 \to 1$ (distance: 12)

21 / 73

- **Trial 1: Divide and conquer**

首先我们从最简单的实例入手，如下图所示，其最短环游非常容易计算（最短环游路线已用红色示出）。



图：旅行商问题的 3 个实例：n = 3, 4, 5，其中结点对之间的距离标注于相应的边上。

接下来我们考察复杂的实例能否分解成简单实例、以及能否使用简单实例的解组合出复杂实例的解。然而不幸的是，虽然一个复杂的 TSP 实例能够比较容易地分解成简单实例，但是用简单实例的解组合出复杂实例的解却不太容易。如上图所示，c 的最优解不等于 a、b 的最优解的组合。

上述困难导致我们难以设计出直接求解 TSP 问题的分而治之算法。因此我们转换一下目标，先求解一个辅助问题：计算从起始结点 s 出发、经过**结点集合 S 中的所有结点**一次且仅一次、最终到达目的结点 x 的最短路径，其长度记为 $M(s, S, x)$ 。最终再从 x 回到 s，即

$$TSP = M(s, S, x) + d_{xs}$$

分解思路：每次从 S 中拿出一个元素。

此即 Held-Karp algorithm [1962]，步骤如下

function $\mathrm{TSP}(D)$

1: **return** $\min_{e \in V, e \neq s} M(s, V - \{e\}, e) + d_{es}$;

function $\mathrm{M}(s, S, e)$

1: **if** $S = \{v\}$ **then**
2: $\mathrm{M}(s, S, e) = d_{sv} + d_{ve}$;
3: **return** $\mathrm{M}(s, S, e)$;
4: **end if**
5: **return** $\min_{i \in S,\ i \neq e} \mathrm{M}(s, S - \{i\}, i) + d_{ei}$;

- **Trial 2: Improvement strategy**

不考虑复杂度，最容易想到的一种基于改进的策略是暴力法迭代改进（指数级搜索空间）。

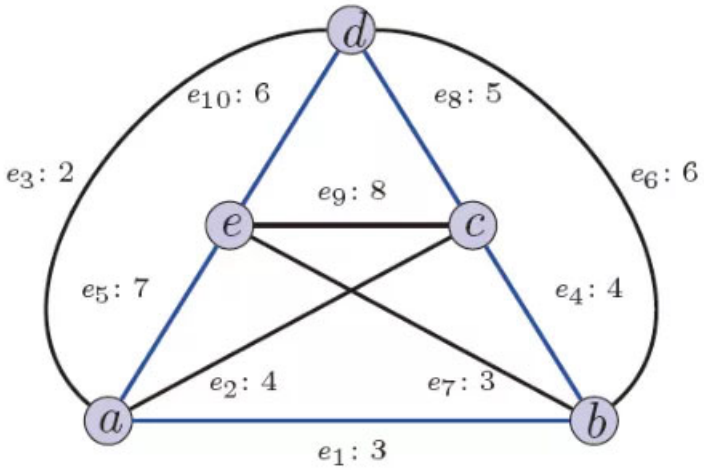
function GENERICIMPROVEMENT(V, D)

```
1: Let s be an initial tour;
2: while TRUE do
3:    Select a new tour s' from the neighbourhood of s;
4:    if s' is shorter than s then
5:       s = s';
6:    end if
7:    if stopping(s) then
8:       return s;
9:    end if
10: end while
```

Here, **neighbourhood** is introduced to describe how to change an existing tour into a new one;

- **Trial 3: "Intelligent" enumeration strategy**

同样的例子，

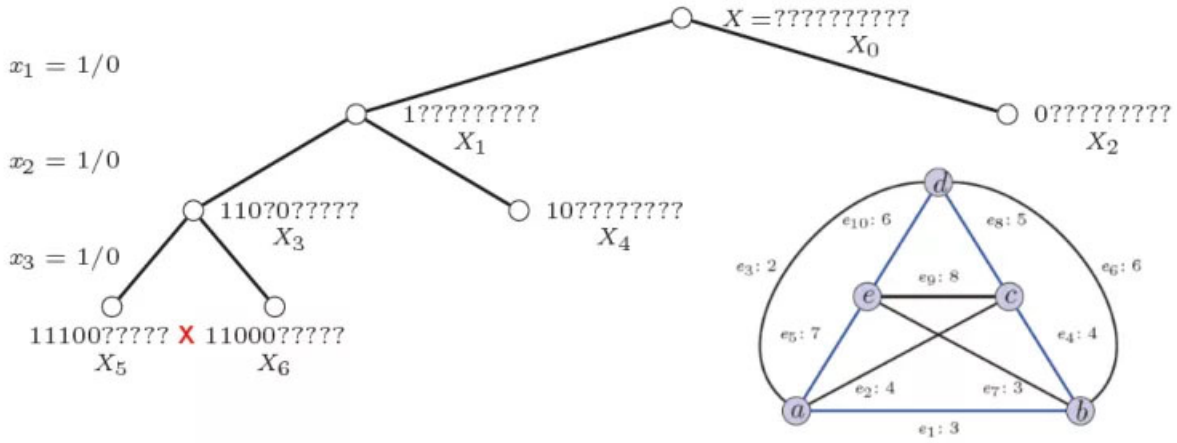


注意到，问题的解可以用 0/1 向量表示，如最优解（对应上图中的蓝线）可以表示为：

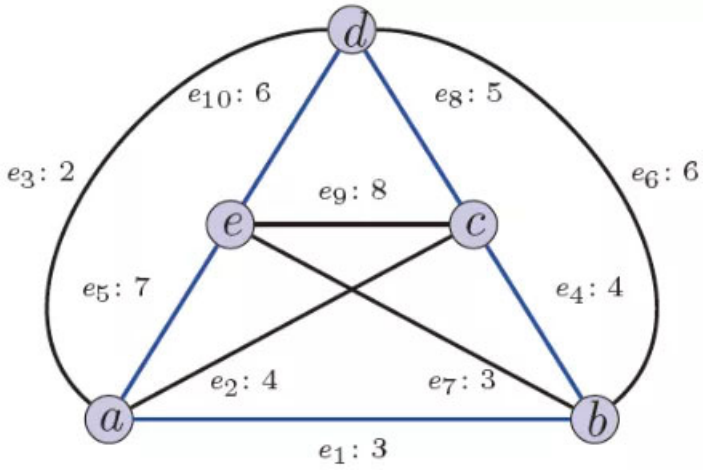
$$X = [1; 0; 0; 1; 1; 0; 0; 1; 0; 1].$$

其中， x_1 表示 e_1 不为 0.

如下，我们可以用类似于树的搜索（表示枚举的过程）



可以看到， X_3 处不是 11??....，而是 110?....，为什么第 3 位也确定了？因为环游过程中 node a 的度不能大于 2.
上面这种枚举方法是没有效率的，下面我们通过启发式的剪枝方法加快枚举的过程。
所谓启发式算法是指，不能证明、但在工程实践中 work 的一类算法。
下面我们通过设置 lower bound function 完成剪枝，以加快算法过程。
lower bound 如何确定呢？有以下“先验”：所有顶点分别的最短的两条邻边之和，总是小于或等于两倍的最优环游路径。（For each city, we select the shortest two adjacent edges; The sum of these 2n edges is less than or equal to 2 times the optimal tour.）



因此，对于上图，其 lower bound 为

$$\frac{1}{2}[(2+3)+(3+3)+(4+4)+(2+5)+(3+6)] = 17.5$$

从而，比方说 X = 10????????，那么其 lower bound 为

$$\frac{1}{2}(5+6+9+7+9) = 18$$

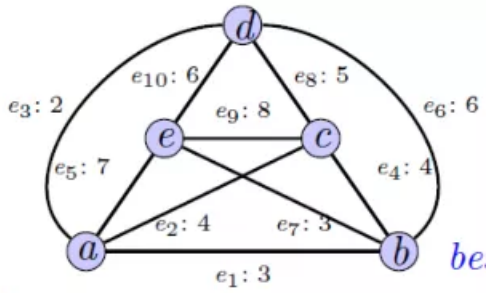
算法流程为

function IntelligentBacktrack

1: Let $A = \{X_0\}$. // Start with the root node X_0 =??...?. Here A denotes the active set of nodes that are unexplored.
2: $best\_so\_far = \infty$;
3: **while** $A \neq NULL$ **do**
4: **Choose** a node $X \in A$ **with lower bound less than** $best\_so\_far$, and remove X from A;
5: Select an undetermined item in X, numerate this item, and thus **expand** X into nodes $X_1, X_2, ..., X_k$;
6: **for** $i = 1$ to k **do**
7: **if** X_i represents a complete solution **then**
8: Update $best\_so\_far$ if X_i has better objective function value;
9: **else**
10: **if lowerbound**$(X_i) \leq best\_so\_far$ **then**
11: Insert X_i into A; //X_i needs to be explored further
12: **end if**
13: **end if**
14: **end for**
15: **end while**
16: **return** $best\_so\_far$;

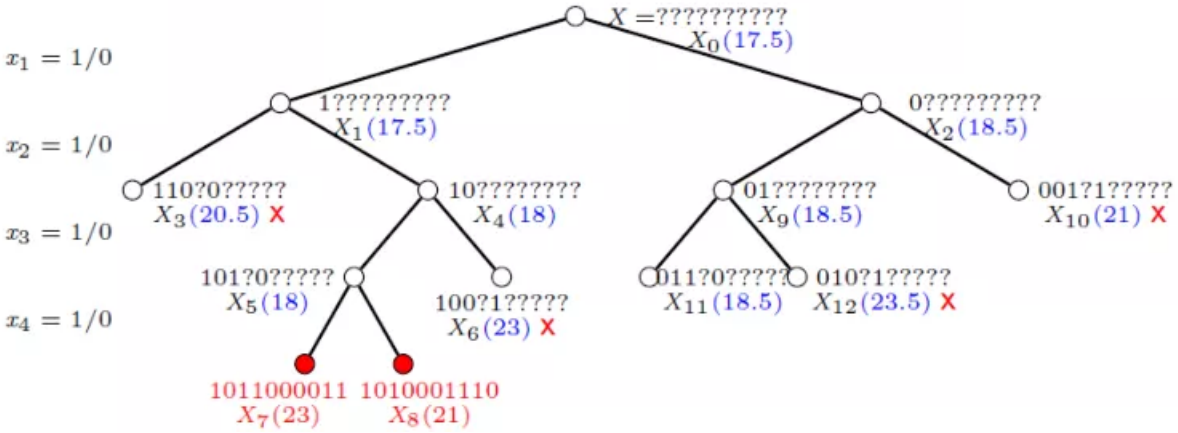
52

从而在下面的实例中，一些枝没有继续 expand，



$A = \{X_{11}\}$
$best\_so\_far = 21$

Partial Solution Tree



5 小结

我们简单回顾了卜老师在第一次课讲的部分内容，并做了简单整理。因篇幅有限，部分内容没有展开。若有错漏，恳请指出。

喜欢此内容的人还喜欢

LOA公众号关闭通知

LOA算法学习笔记

明式家具：文人书房造就的人文雅器

尚艺木工坊

爱奇艺史上规模最大裁员：有的工作室消失，有的部门裁一半人！龚宇上月曾称“开源节流是现阶段重点”，爱奇艺回应

商学院