目录

第一章　深度优先搜索和广度优先搜索

我们以前接触过这两位, 是在学习树的时候.
与其说是两种策略, 不如说是使用了两种数据结构. 深度优先是栈, 而广度优先是队列.

1.1　BFS

下面给出 BFS 的框架:

```
1  // 构造由根组成的队列 Q
2  if Q 中的第一个元素 x 是目标节点
3      stop
4  else
5      从 Q 中删除 x, 将 x 的所有子节点放进 Q 中.
6  if Q == NULL
7      return failed;
8  else goto 2
```

虽然使用了 goto 但也算是简洁了, 其实用 while 也很简洁. 我们说这里的关键就是队列, 其特点就是先进先出. 对于节点 x , 我们将其子节点放入队列, 这些子节点都搞定之后才会继续搞其他的子节点.

1.2　DFS

区别在于使用了栈.

```
1  // 构造由根组成的栈 S
2  if S 中的第一个元素 x 是目标节点
3      stop
4  else
5      Pop (S) ;
6      Push (s); // 将 S 的子节点压进栈.
7  if S == NULL
8      return failed;
9  else goto 2
```

第二章　搜索的优化

2.1　爬山法

基本思想: 在 DFS 的过程中, 使用贪心法确定搜索的方向. 爬山策略使用 “启发式测度” [1] 来排序节点优先程度.

这里笔者拒绝使用 “启发式测度”, 这个听起来很几把无语的词语. 因为这个和测度毫无关系.

我们通过 f 来测量某一个节点, 我们说, 越接近某一值 a 则该节点距离正确答案就越近, 可以按照这一标准来排序节点.

将当前节点 x 的子节点压进栈的时候, 按照上面给出的排列顺序压进栈.

于是乎, 我们的算法思路就是这样:

```
1  // 构造由根组成的栈 S
2  if S 中的第一个元素 x 是目标节点
3      stop
4  else
5      Pop (S) ;
6      Push (s); // 将 x 的子节点按照给出的排列顺序压入栈.
7  if S == NULL
8      return failed;
9  else goto 2
```

我们能够明显地看出, 这个搜索策略并不一定是最优的解, 因为这只是一个贪心选择, 我们的问题不一定满足贪心选择性.

接下来我们要修改的话, 就是考虑到全局的节点.

2.2　Best-First 搜索

类似的, 我们有给出一个函数 f 来考察节点的优先程度.

基本思想:

- 结合深度优先和广度优先的优点
- 根据函数 f 在目前产生的所有节点中选择具有最优的节点进行扩展 [2]
- 选择了全局最优解, 而爬山只是局部最优.

[1] 什么几把名字

[2] 扩展又是什么意思? 我不是说故意挑刺, 但是你不能总是突然冒出一个自己编纂的词语, 然后又不解释是什么意思

```
1 依照 f 的值为比较的key, 构造出一个堆, 先是构造只有起点 r 的堆
2 if H 的根 x 是目标节点 then stop;
3 delete_max(H); 将 x 的子节点插入heap;
4 if H == NULL then failed;
5 else goto 2
```

2.3 分支界限

TODO 基本思想:

- 上述方法很难用于求解优化问题 [3]
- 分支界限策略可以有效地求解组合优化问题.
- 发现优化解的一个界限, 缩小解空间, 提高求解的效率 [4]

我们这里的例子是这样的

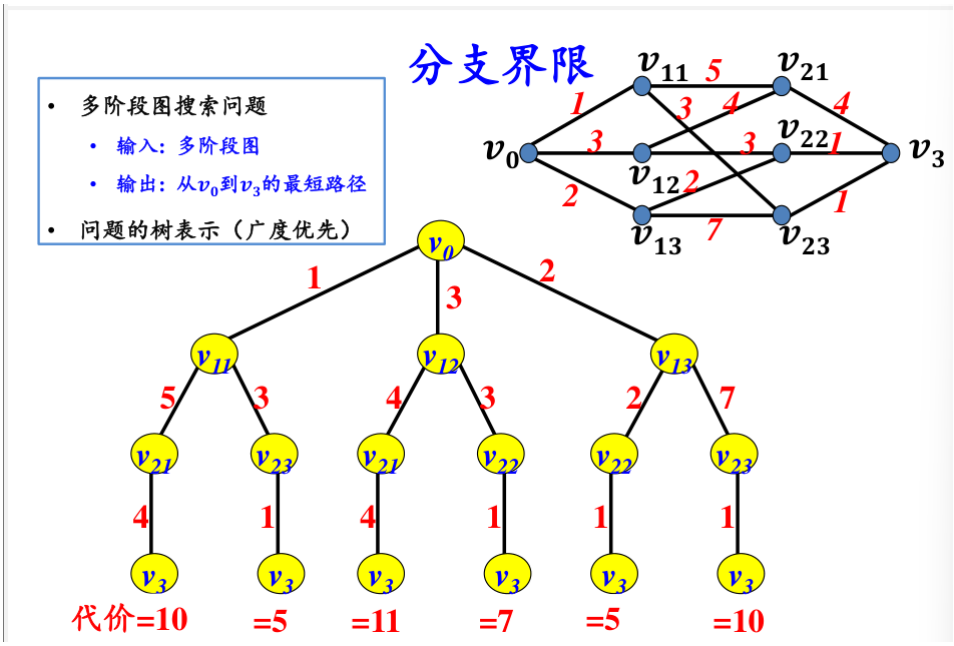

图 2.1: 例子

1 使用爬山策略, 找出一个路径 (选择权重最小的边), 1 3 1 这条路

2 根据这个解, 进行剪枝. 已知这个路径的长度是 5 , 我们将上一层节点中大于 5 的节点剪去. 这就是缩小了解的范围.

我的评价: 这讲的什么几把, 指 ppt
剪完了然后呢? 难道是根据 best-first 再搜索一边吗? 搞不懂捏.

[3] 为什么
[4] 你在说什么

第三章　剪枝方法与人员安排问题

3.1　问题定义

input : P for person, $P = \{P_1, \cdots, P_n\}$ 是 n 个人的集合.

人员安排的问题之定义实际上是这样, 我们有一个工作的集合, 他是一个偏序集, 我们将其进行一个拓扑排序, 比如说 $\{J_{i_k}\}$ 这是排序中的第 k 个工作, 意思就是这个工作由第 k 个人来担任.

同时给出了一个矩阵 (X_{ij}) , X_{ij} 的意思是, 第 i 个人分配了第 j 个工作所需要的时间.

例. 给定$P = \{P_1, P_2, P_3\}$，$J = \{J_1, J_2, J_3\}$，$J_1 \leq J_3$，$J_2 \leq J_3$。

$P_1 \to J_1$、$P_2 \to J_2$、$P_3 \to J_3$是可行解。

$P_1 \to J_1$、$P_2 \to J_3$、$P_3 \to J_2$不可能是解。

图 3.1: 一个例子

于是说, 我们的目的就是要遍历所有的拓扑排序, 从中找到一个最优解, 总体来说这是一个遍历的过程. 我们可以回想以前是怎么找拓扑排序的: 每次选择一个 “没有前序元素的” 的元素, 作为当前根节点的子节点. 对于这些获得的子节点也是使用这样方法, 也就是递归地处理子节点.

1. 生成空树根
2. 选择偏序集中没有前序元素的元素, 作为当前根节点的子节点
3. For root 的每一个子节点, do
4. S = S - v
5. 将 v 作为根, 递归地处理 S

我们这就生成了一个拓扑排序对应的树, 这个树上, 从根节点到叶子的一条路就是一个拓扑排序. 我们当然可以对这棵树上的搜索进行优化.

3.2　算法的优化: 针对代价矩阵做出的优化

命题 1. *将代价矩阵的一行或者一列, 减去同一个数字, 不影响优化解的求解.*

- 代价矩阵的每行减去同一个数, 使得每一行以及每一列是找有一个零, 其余元素非负.
- 减数的和是解的一个下界.

优化之后, 根节点的值为前面计算的下界.

总之, 通过分支界限法, 来剪枝, 优化之后的树能够有效的进行剪枝

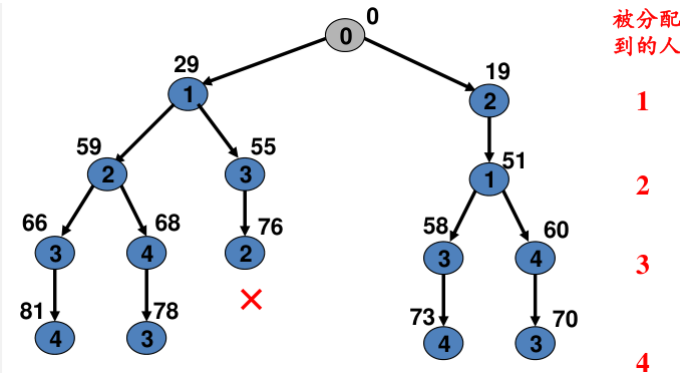

图 3.2: 优化前的树

- 解空间的加权树表示

$$\begin{bmatrix} 17 & 4 & 5 & 0 \\ 6 & 1 & 0 & 2 \\ 0 & 15 & 4 & 6 \\ 8 & 0 & 0 & 5 \end{bmatrix}$$

优化代价矩阵

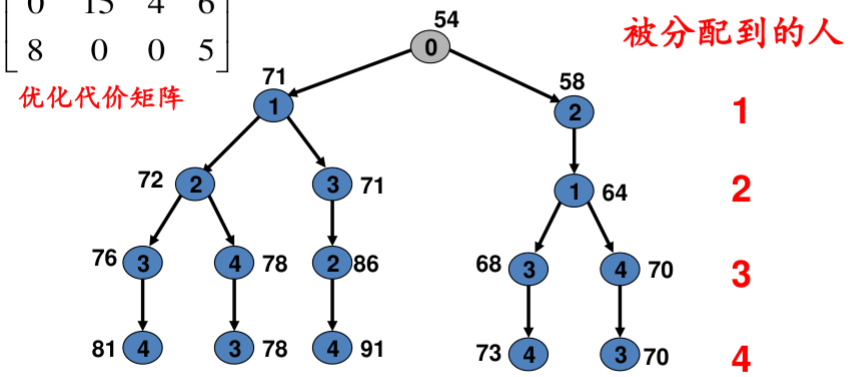

图 3.3: 优化后的树

- **分支界限搜索（使用爬山法）算法**
  1. 建立根节点，其权值为解代价下界；
  2. 使用爬山法，类似于拓朴排序序列树生成算法求解问题，每产生一个节点，其权值为加工后的代价矩阵对应元素加其父节点权值；
  3. 一旦发现一个可能解，将其代价作为界限，循环地进行分支界限搜索：剪掉不能导致优化解的解，使用爬山法继续扩展新增节点，直至发现优化解。

第四章　旅行商问题

todo
我是想问, 你能不能去吃粪?

第五章　A* 算法

todo