动态规划：搜索空间的状态合并

蒋炎岩 <jyy@nju.edu.cn>

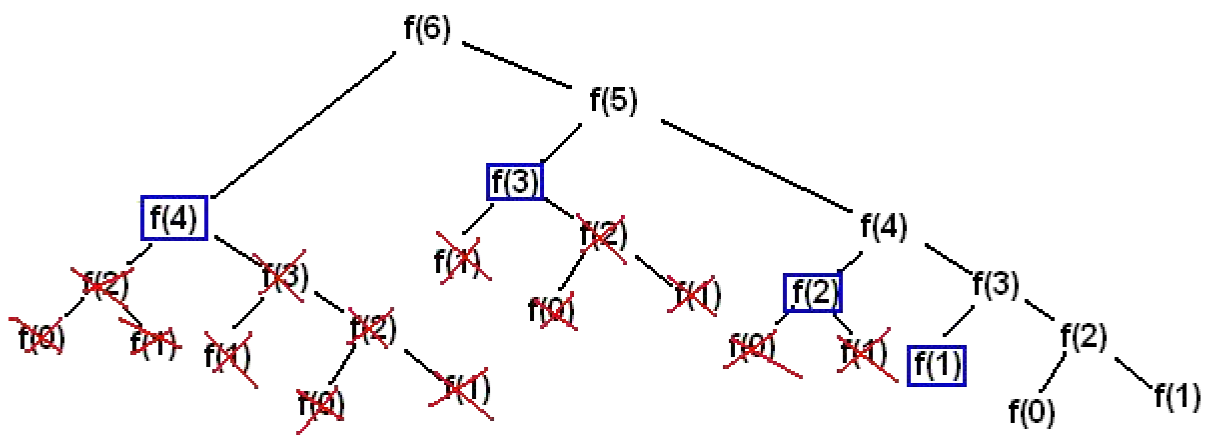
南京大学 | 计算机软件研究所 | 系统与软件分析研究组

System & Program Analysis Research

我们熟悉的动态规划

- 想要解决一个问题，求$f(P)$
  - 把$f(P)$分解成若干个小问题$f(P_1)$, $f(P_2)$, ...分别求解
  - 然后再算出$f(P)$的数值
  - 每个子问题只需求解一次

- 这里的关键：问题如何分解？
  - 用合适的状态表示建立问题和子问题(难点)



动态规划：另一个视角

复习：枚举法

- 针对离散型的最优化或计数问题
  - 理解“最优化”或“计数”的空间 → 搜索空间
  - 枚举所有可能性，进行检查

- 例子
  - 枚举$1,2,\ldots,n$的全排列
  - 枚举$\{1,2,\ldots,n\}$的所有子集
  - 枚举图中的所有路径(dfs)

练习：建立搜索空间

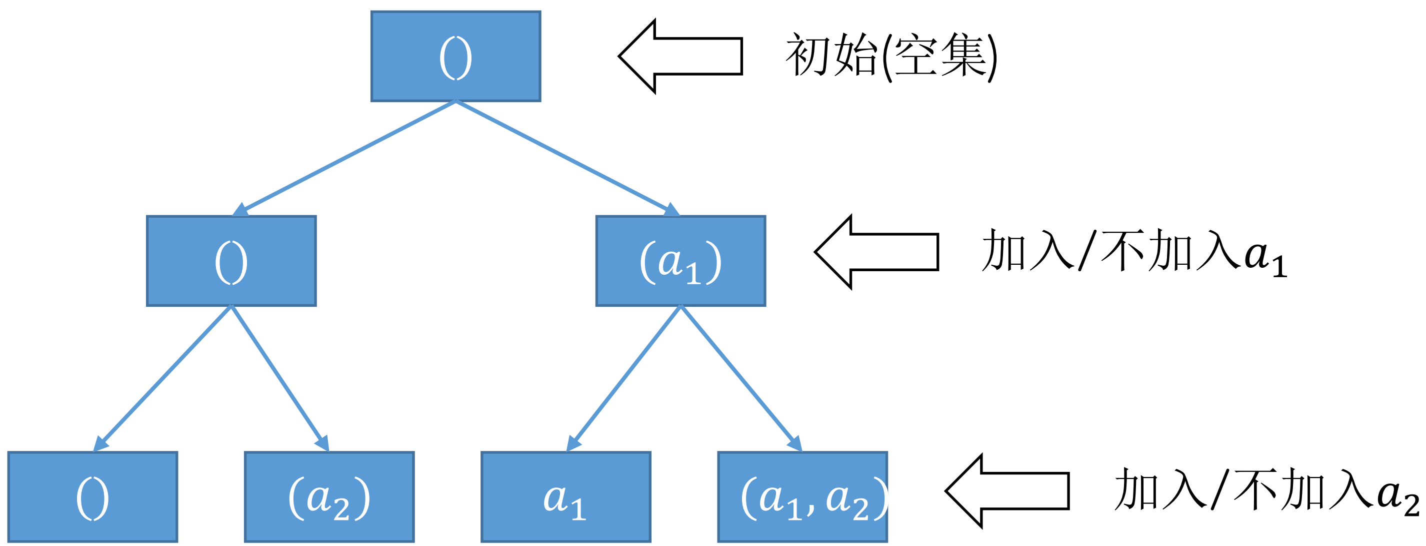
- (LIS) 最长上升子序列
  - 给数组$a_1, a_2, \dots, a_n$，求其中最长的下标、数值都上升的子序列

- (LCS) 最长公共子序列
  - 给两个字符串s_1, s_2，求最长的字符串s，其能通过s_1和s_2删除若干字符得到

- (Hamilton) 哈密顿路径
  - 给一个有向图$G(V, E)$和权值W，求s到t权值最小的路径，其经过每个点一次且仅一次

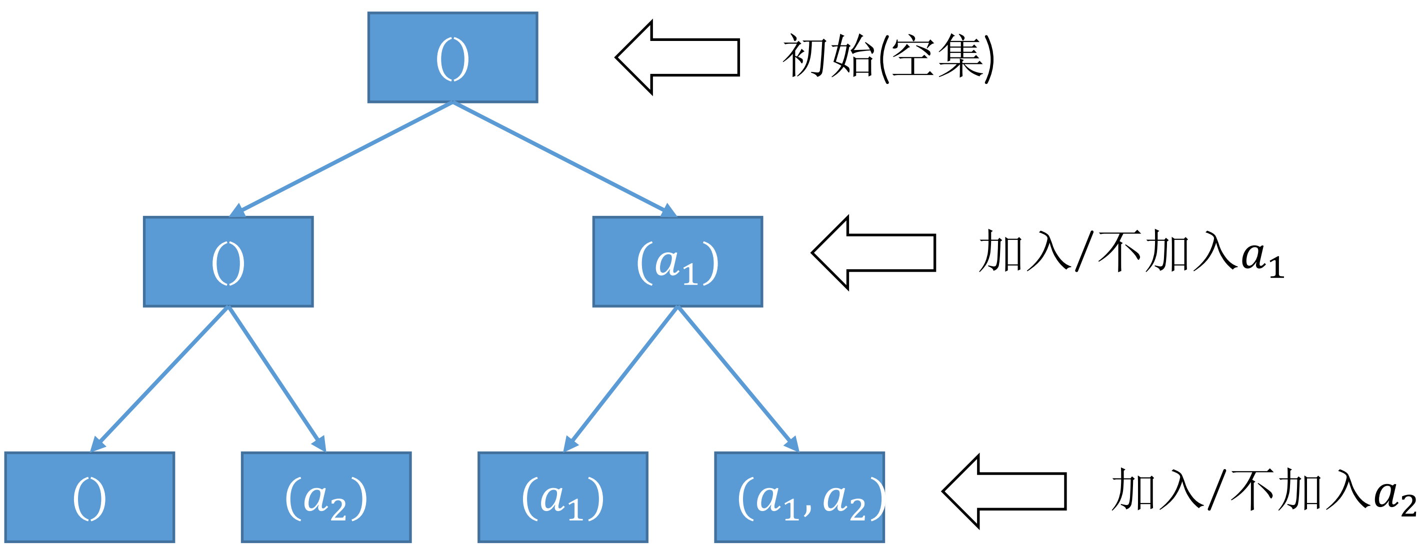
搜索空间：遍历

- 搜索空间通常有一个图结构，通过图中的边可以算出节点(状态)上的目标函数值
  - 以下是dfs生成所有子集的过程(可以用于求解LIS)
  - 类似可以生成所有子序列、所有路径……



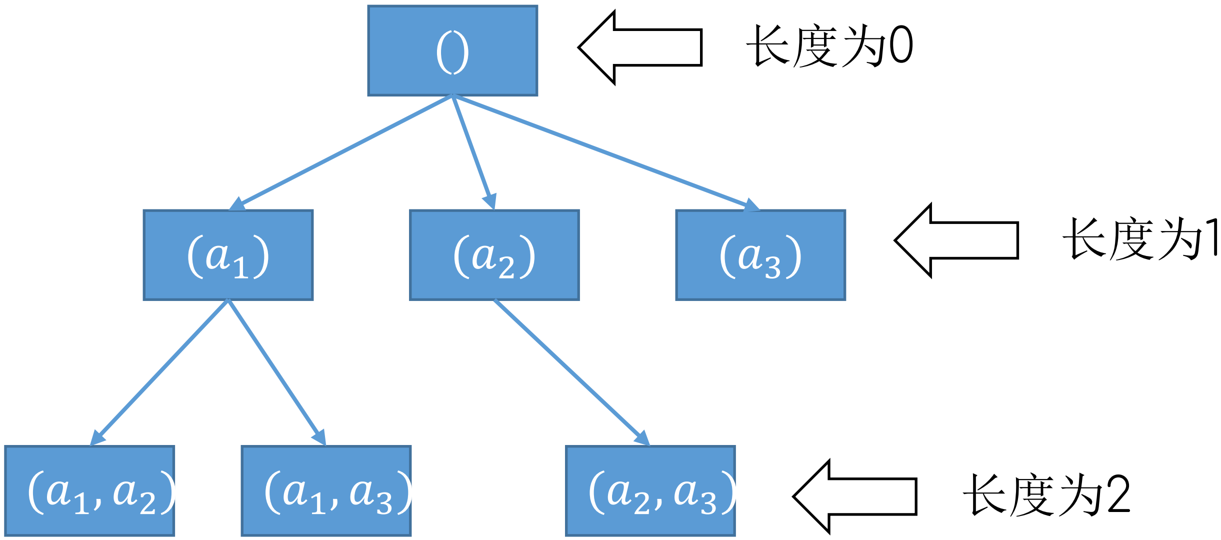
LIS：搜索空间

- 生成所有子集，按顺序考虑编号为i的元素是否要放入集合中
  - 删除不合法的状态，大小最大的子集就是LIS

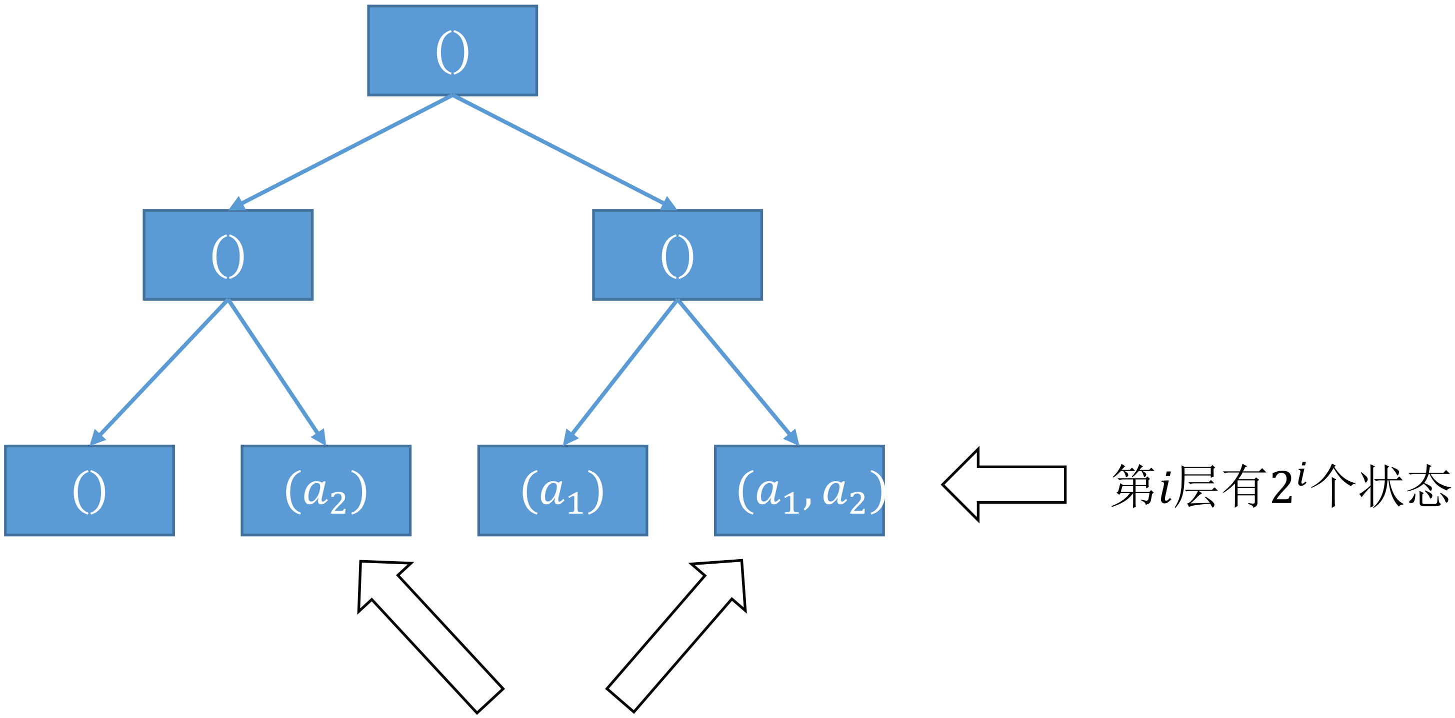


LIS：另一个搜索空间

- 同样生成所有子集，但每次选一个下标更大的元素加入
  - 删除不合法的状态，大小最大的子集就是LIS



LIS：避免重复计算

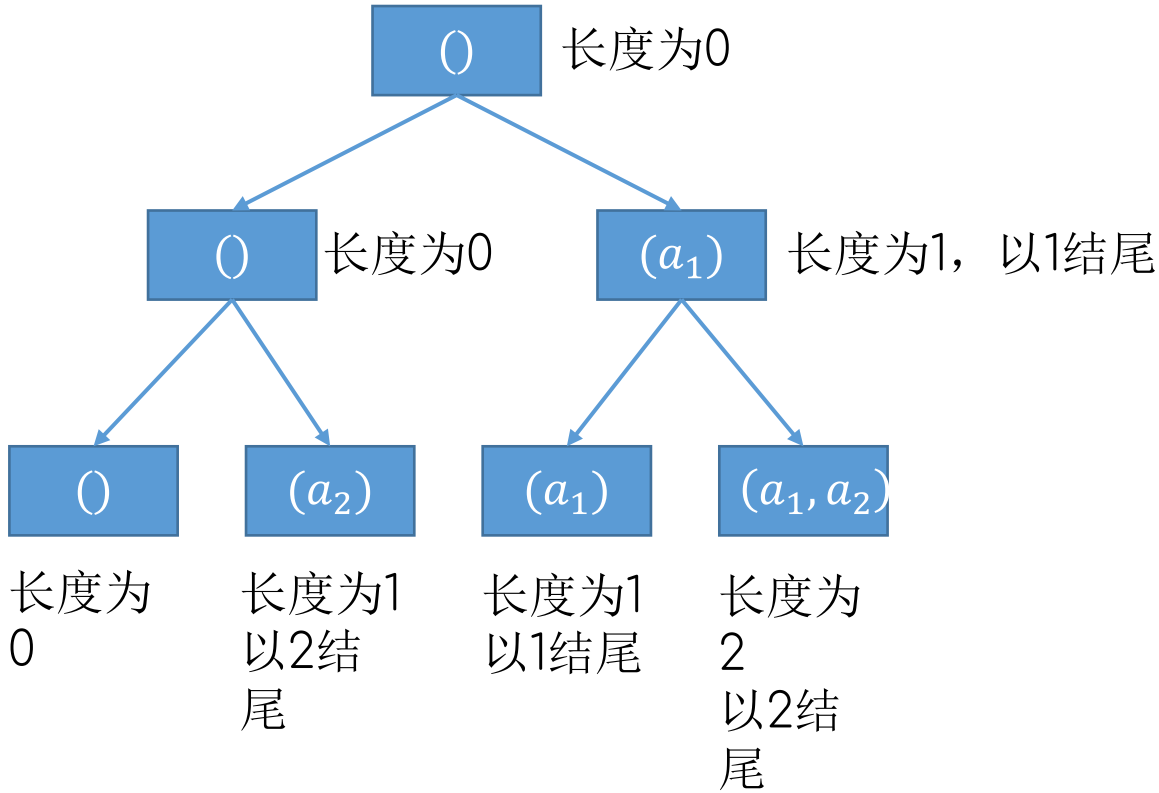


有些状态明显是重复的

如果{1,2,3} {1,3} {2,3} {3}都合法 → 只要保留{1,2,3}即可

如果只有{1,3} {2,3} {3}合法 → 只要保留{1,3}即可

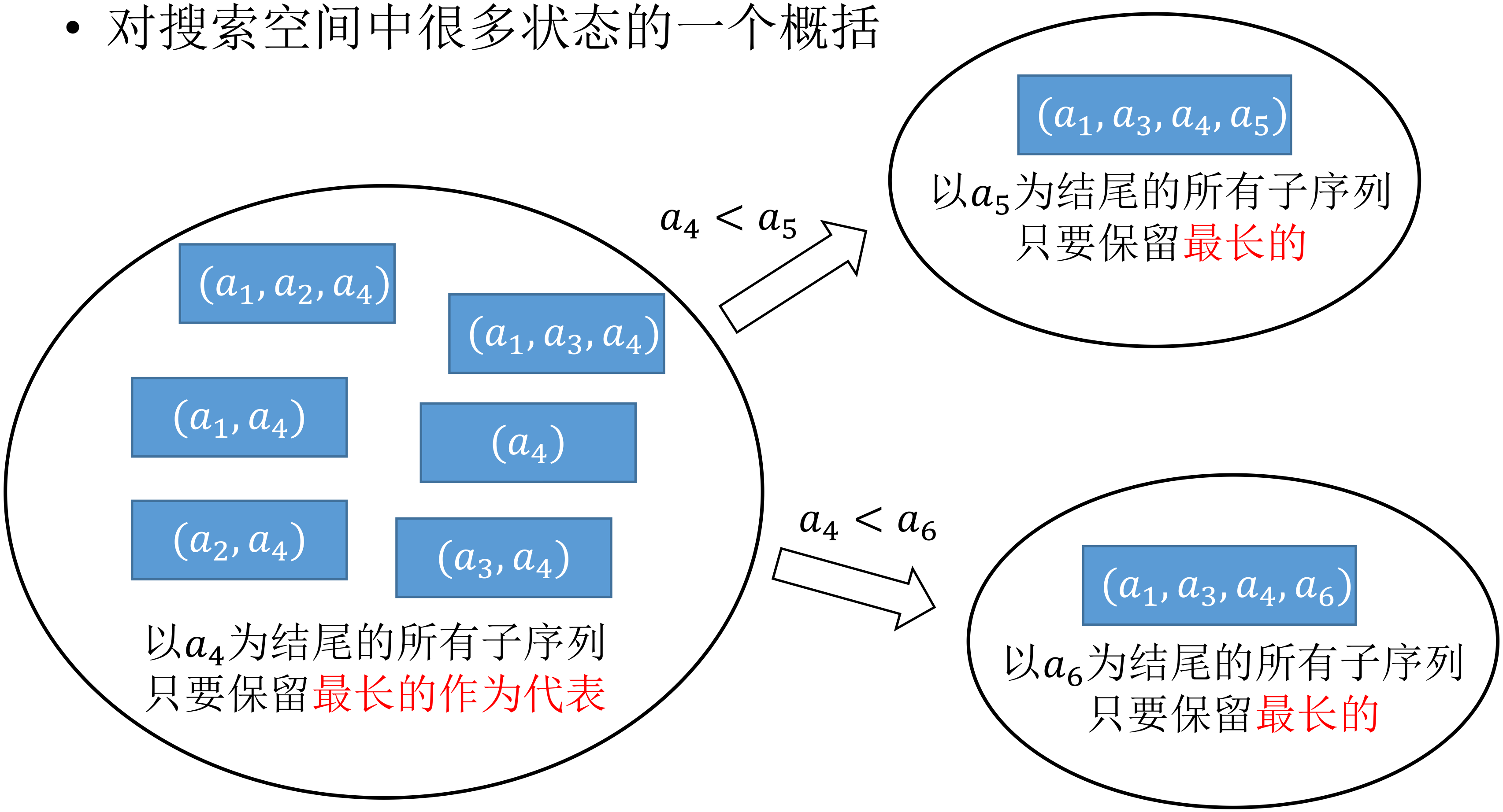
LIS：避免重复计算 (cont'd)



- 在某一轮，只需要确定以每个数字结尾的子序列，最长的子序列是多少

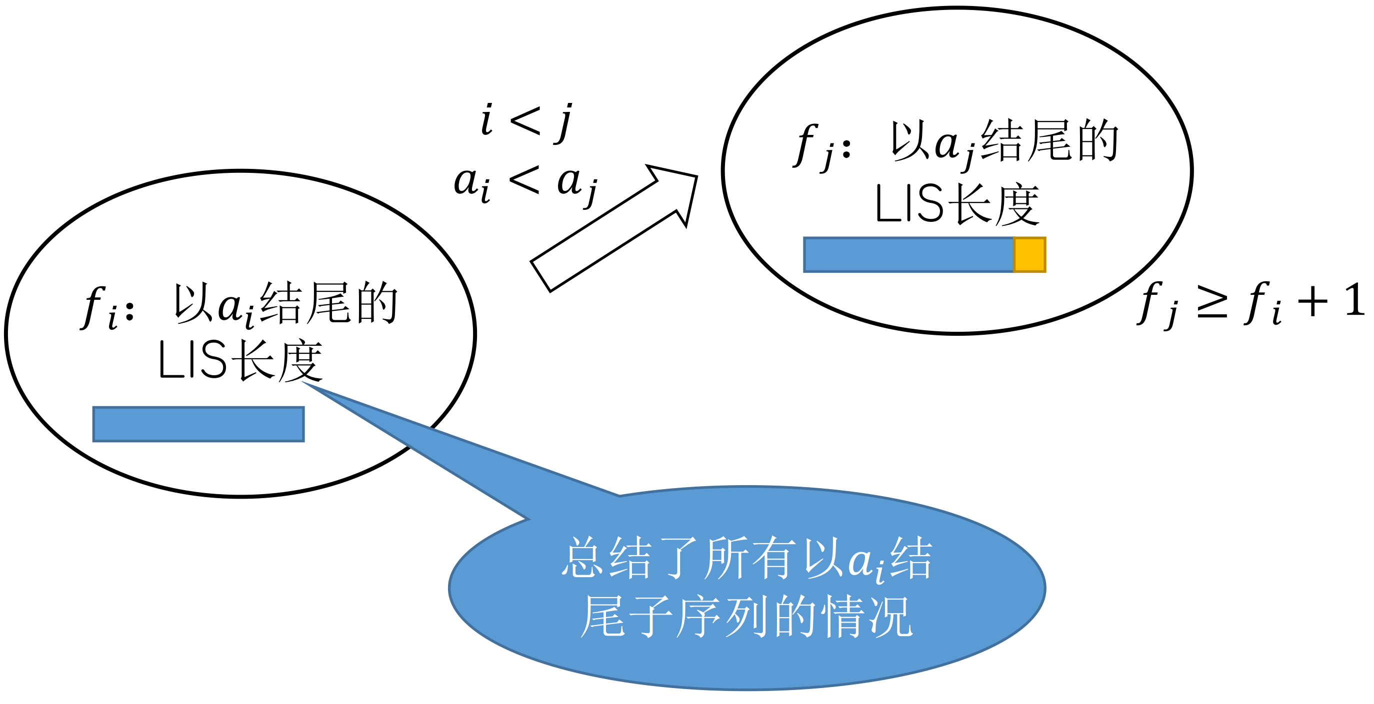
动态规划：搜索空间的“折叠”

- 对搜索空间中很多状态的一个概括



动态规划：对搜索空间的概括

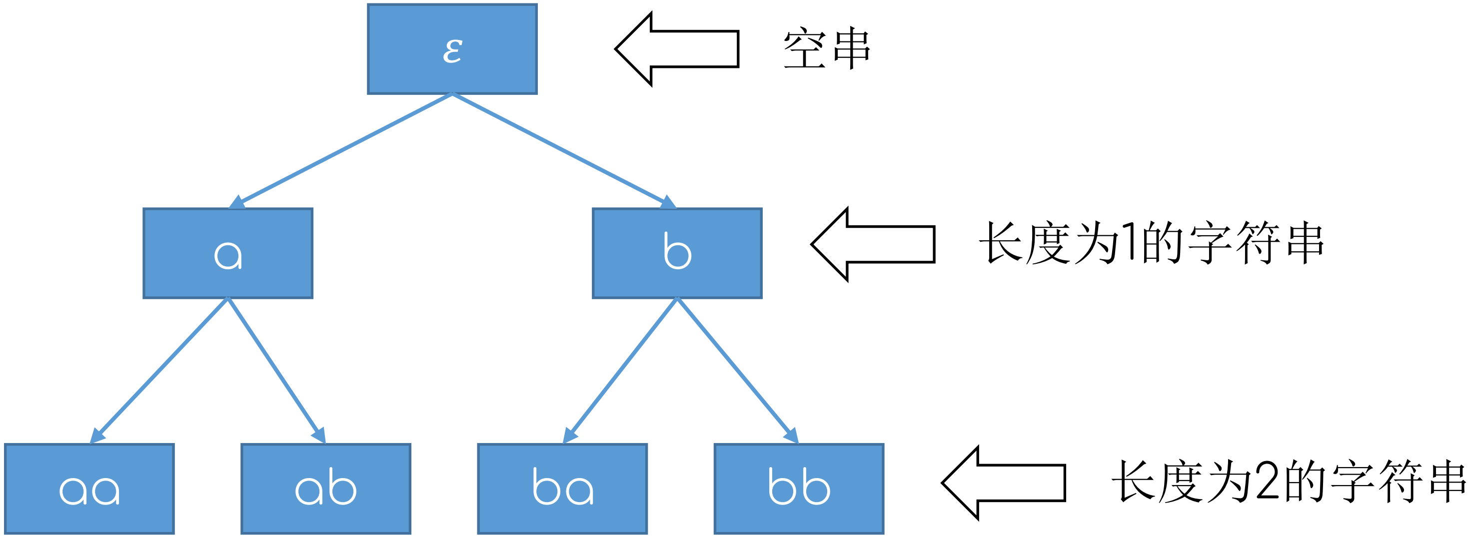
- 把所有可能的状态分组，并做出对问题求解有效的概括
  - 有些搜索空间比较适合dp，有些就比较糟糕



另一个例子：LCS

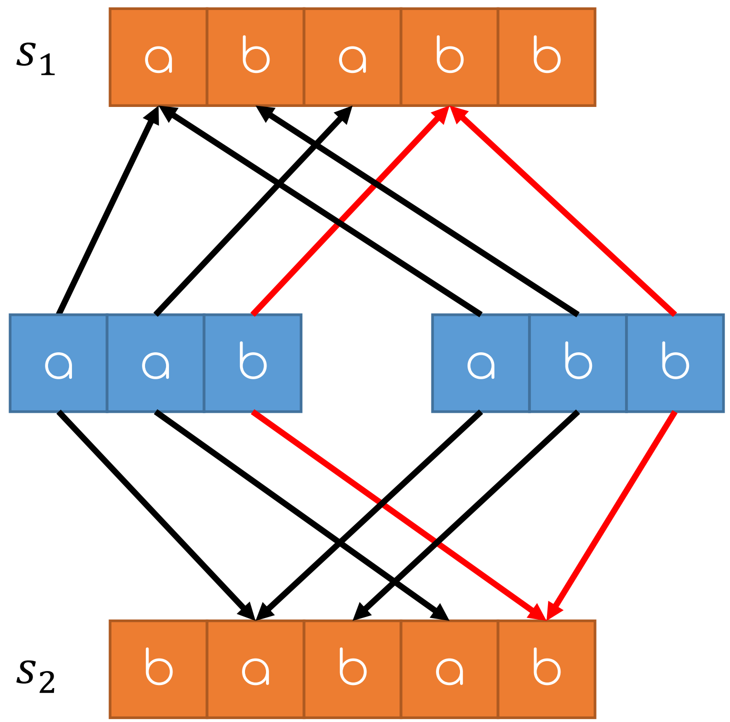
动态规划：避免重复计算

- (LCS) 最长公共子序列
  - 给两个字符串s_1, s_2，求最长的字符串s，其能通过s_1和s_2删除若干字符得到

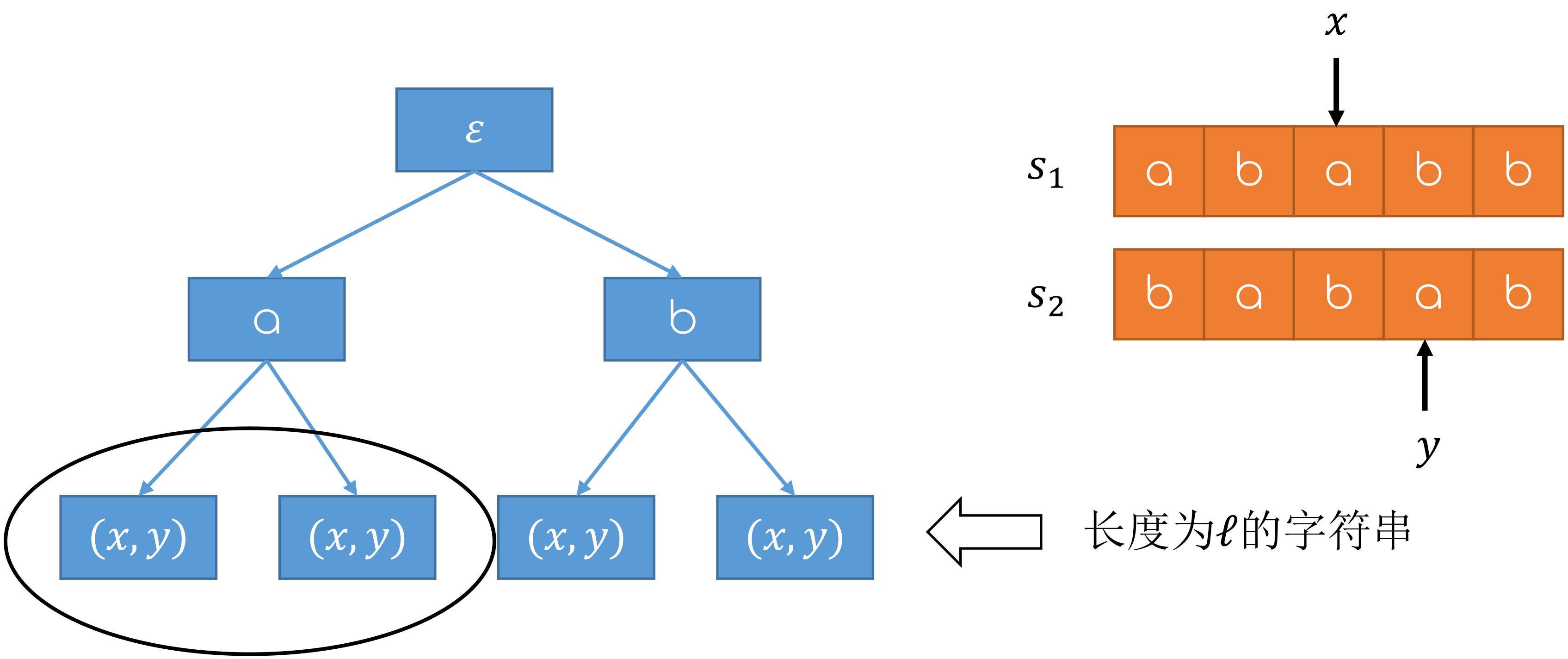


例子

- 从"增加一个字符"的角度，aab和abb是完全等效的
  - 只要考虑最后一个字符在s_1, s_2中的位置



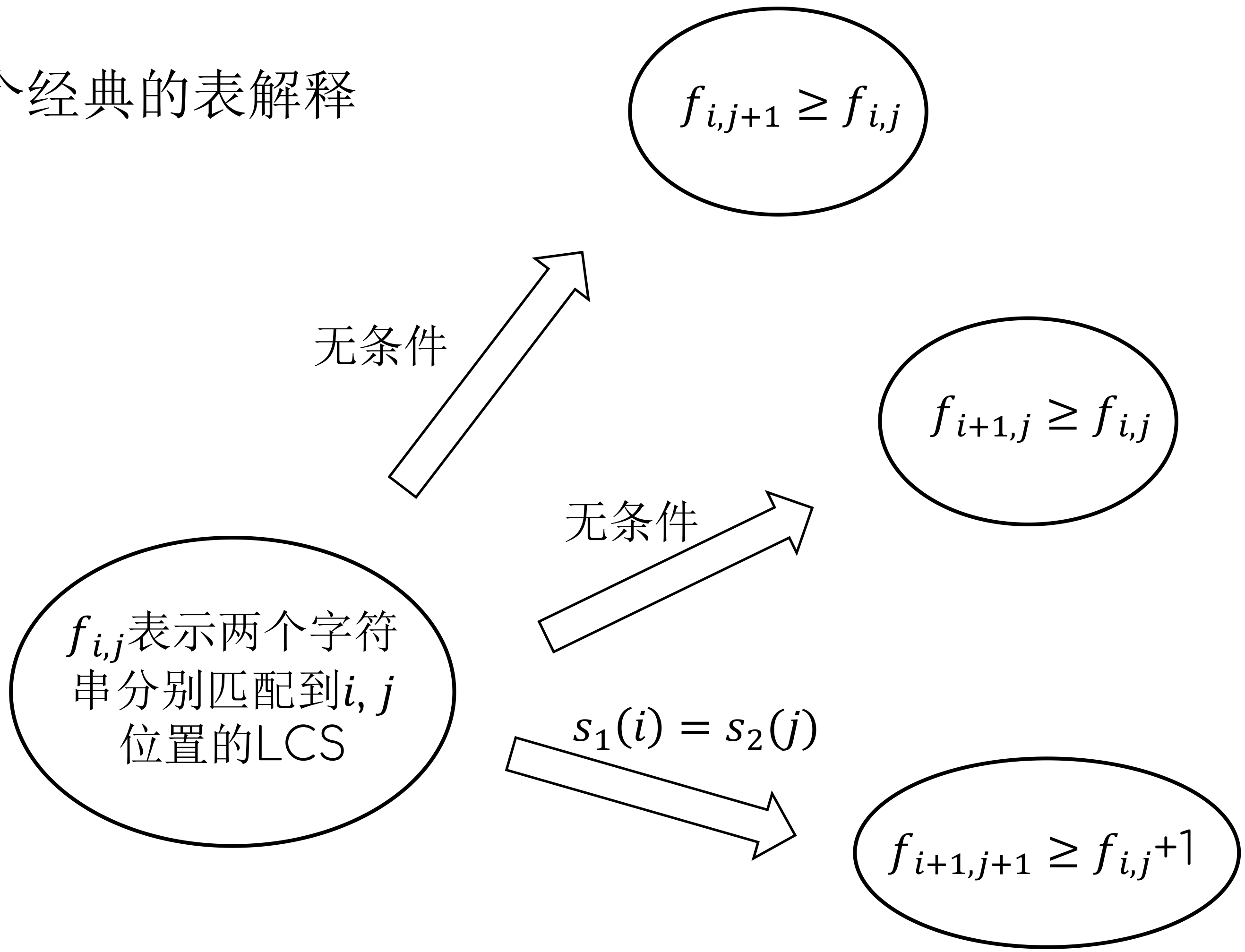
状态空间：概括



长度为3，两个字符串匹配位置分别是(x,y)的只需要保留一个

LCS：经典DP

- 有一个经典的表解释



路径：Bellman-Ford和Floyed-Warshell

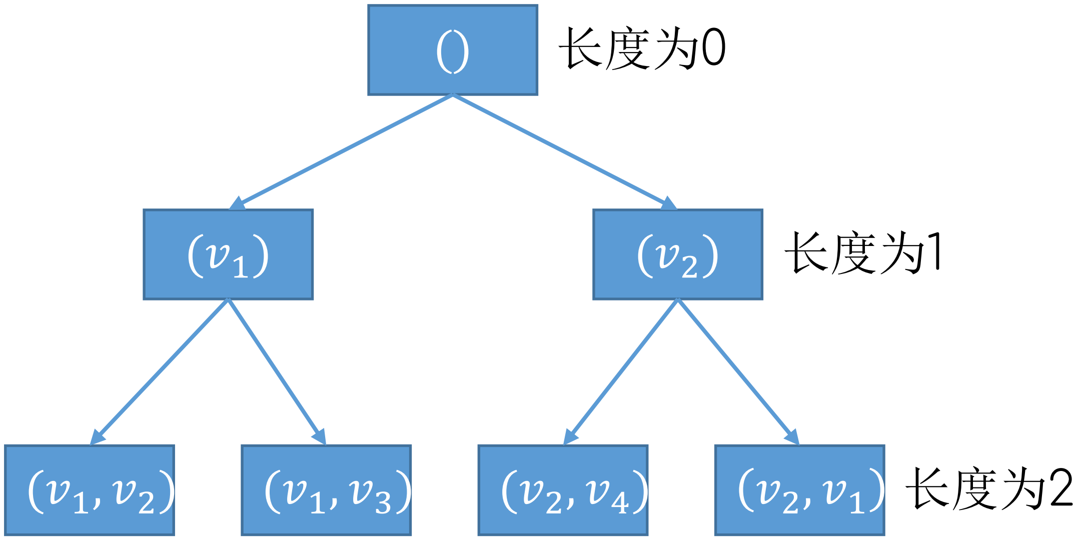
Bellman-Ford：大家的第一个最短路算法

- 求图中s到t的最短路径

```
for (int i = 0; i < n; i++)
  for (int j = 0; j < m; j++) {
    int u = E[j].from, v = E[j].to, w = E[j].weight;
    if (dist[u] + w < dist[v]) {
      dist[v] = dist[u] + w;
    }
  }
```

理解Bellman-Ford：搜索空间

- 如何生成所有路径？



- 对于同样结尾的路径，只要保留最短的就行了
  - $f_{i,j}$表示长度为i路径，位于j点的最短路/路径数量/...

更难的算法：Floyd-Warshell算法

- 非常简单的三重循环(k, i, j)
  - 为什么它是对的？

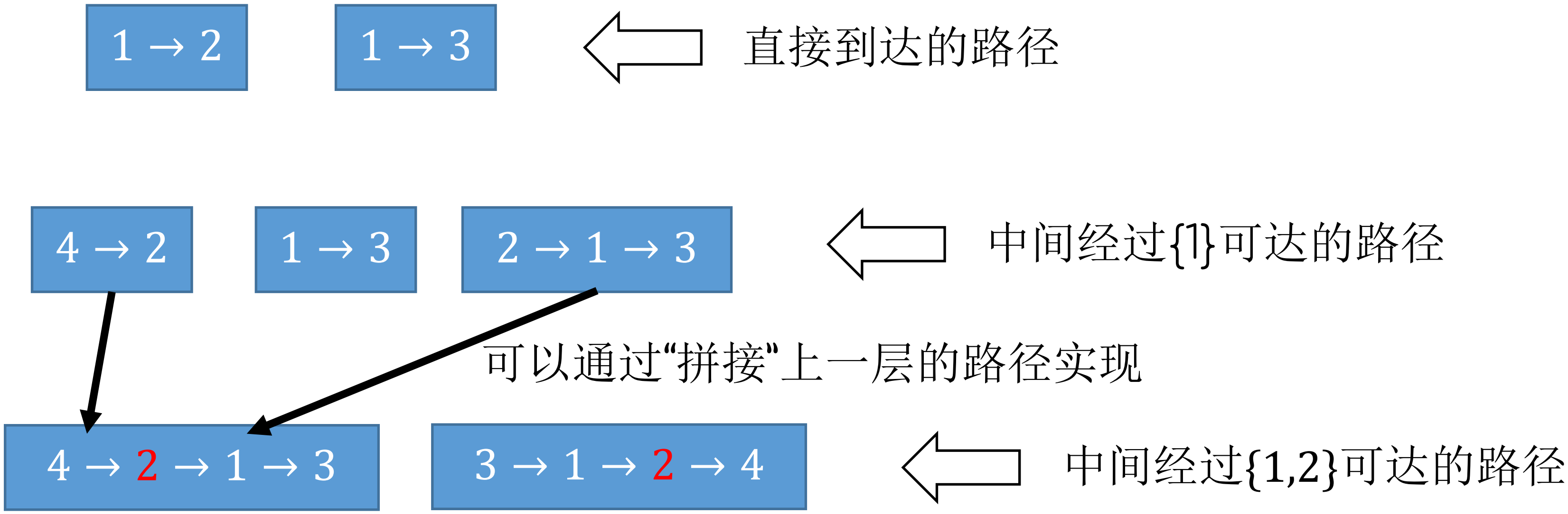
```
void floyd_warshell() {
  for (int k = 0; k < n; k++)
  for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++) {
    weight_t d = dist[i][k] + dist[k][j];
    if (dist[i][j] > d) dist[i][j] = d;
  }
}
```

更难定义的搜索空间

- 所有点对之间的所有简单路径
  - $1 \to 2 \to 4 \to 3$
  - $4 \to 1 \to 2$
  - $3 \to 4$
- 不同于以往我们经常处理的对象
  - 子集、路径……

Floyed-Warshell：搜索空间表示

- 看起来是个有些反直觉的(巧妙)设计



所有$s \to t$路径中，只要保留最短的

Floyed-Warshell：动态规划

$f_k(i,j)$表示经过$\{1,2,\ldots k\}$这些节点的$i \to j$最短路径

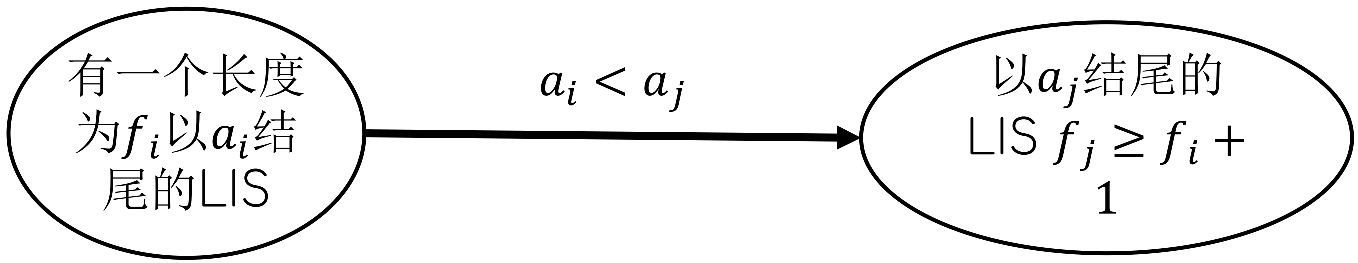
$\Rightarrow$

$f_{k+1}(i,j) \le f_k(i,k) + f_k(k,j)$

动态规划：搜索空间的概括

搜索空间的概括

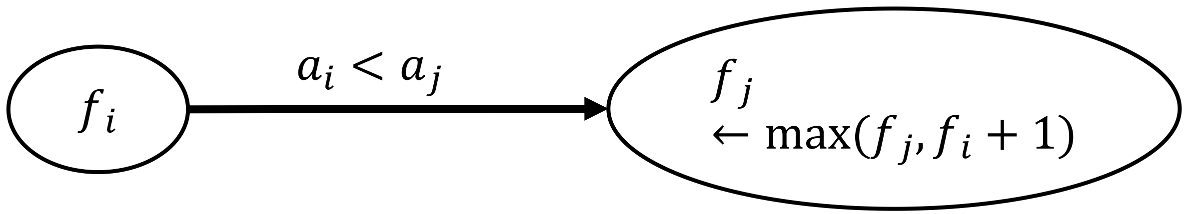
- 找到搜索空间中的冗余 ⇔ 定义动态规划的“状态”



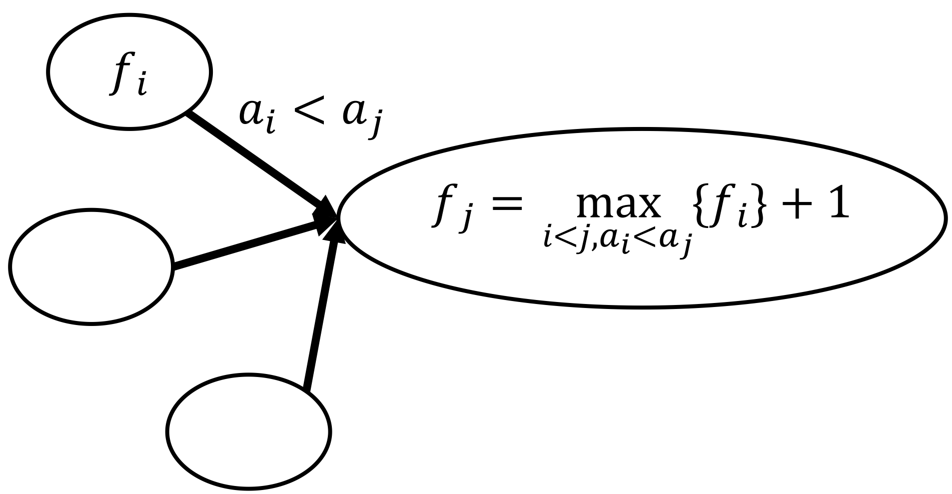
- 用f_i表示搜索空间中很多状态的“概括”
  - 找到合适的f_i表示就已经解决了问题的大半了

动态规划的两种实现

- 正推(更新)：描述“出边”



- 逆推(直接求解方程)：描述“入边”



搜索空间：子集

- 搜索空间：$\{\emptyset, \{a\}, \{b\}, \{a, b\}, \ldots\}$
  - LIS：最长上升子序列
  - 背包：n个物品，每个物品有重量/价值/数量
    - 01背包：每个物品数量恰好有一个，求重量限制下的最大价值
    - 多重背包：每个物品数量给定 → 可以转换为01背包

- 对于$\{1,2, \ldots, n\}$，按顺序考虑$\{1,2, \ldots, i\}$的子集
  - LIS：考虑$a_1, a_2, \ldots, a_i$中所有的子集
  - 背包：考虑$a_1, a_2, \ldots, a_i$中所有的子集

搜索空间：区间中的整数 (数位dp)

- 搜索空间：$\{\ell, \ell+1, \ell+2, \ldots, r\}$ (ℓ, r可能很大)
  - 统计其中二进制数表示中，1比0多的数量；
  - 统计其中既能被13整除，字符串中又不含13的数量；
  - 统计奇数位和-偶数位和恰好为k的和；
  - ……

- 搜索空间的两次分解
  - 分解成412XXXXX的搜索空间(套路)
  - 对搜索空间进行进一步合并

搜索空间：图结构 (状态压缩)

- $s \to t$的最小权哈密顿路径
  - 已知NP-Complete，目前看搜索无法避免
  - 搜索空间中是否有多余？
    - $1 \to 2 \to 3 \to 4 \to \ldots$
    - $1 \to 3 \to 2 \to 4 \to \ldots$

- (NOIP2017 Treasure) 给一个n个点的无向图，求一个生成树
  - 生成树的根可以任意指定
  - 生成树每条边e付出的代价是$w_e \times d$，其中d是e距离根最近节点的深度

搜索空间：树

- (IOI2005 River)
  - 树上每个节点都会产生一定的货品，货品沿树向根运输
  - 单位货品运输单位长度(边有长度)需要单位代价
  - 现在允许增加k个处理站，货品到达处理站就不用继续运输
  - 问如何布置处理站最小化运输代价

- 搜索空间：非根节点中选出k个处理站
  - 所有(x, k)满足在x点放置处理站、子树中有k个处理站，我们只关心运输代价最小的那个
  - 但$f(x_1, k_1)$和$f(x_2, k_2)$之间用另一个背包问题求解

例题：任务调度

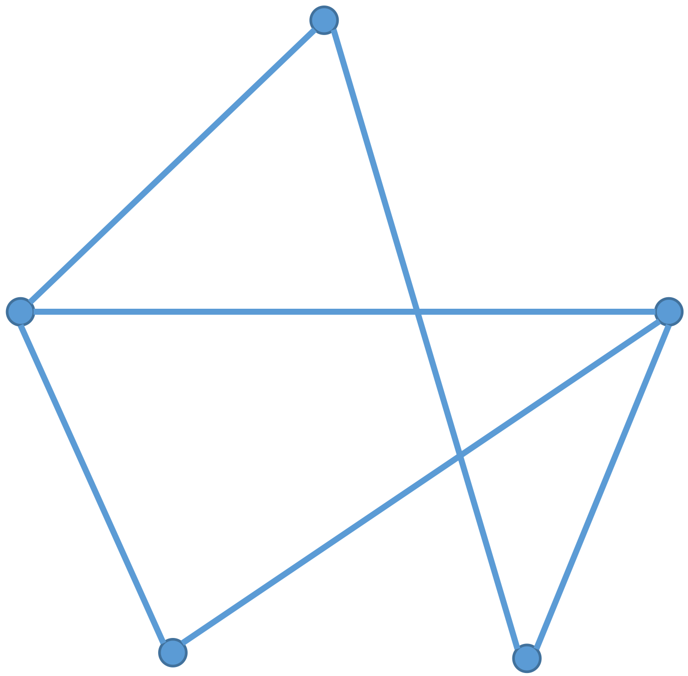
- 有若干任务，第i个任务需要t_i时间执行，且必须在d_i之前完成，获得p_i的收益
  - 问如何选择任务，获得最大收益？

- 和背包的对比
  - 背包：不用考虑过去取的物品的顺序，所以直接考虑每个物品取/不取
  - 这个问题：当前物品的加入会影响之前物品的调度，能否加入是比较复杂的

- 换一个搜索空间，保证物品(任务)能直接加入

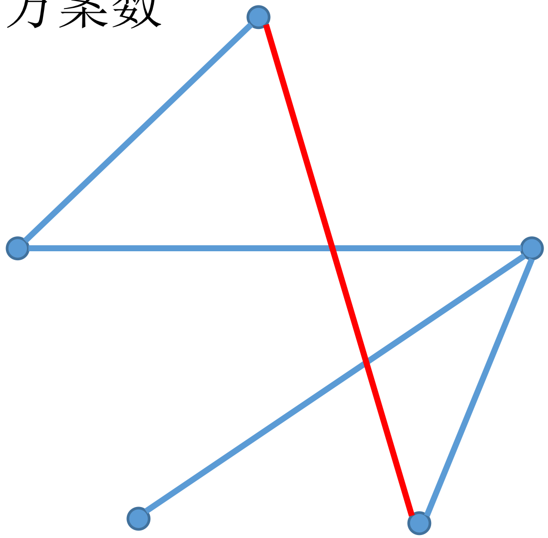
例题：生成树

- 正n边形的顶点有无向边相连，求它生成树的数量，满足任意两条边都没有形内的交点
  - 不是生成树计数(Kirchhoff's Matrix-Tree Theorem)
  - 很多失败的尝试(生成树很难干净地枚举)……



考虑加边的顺序

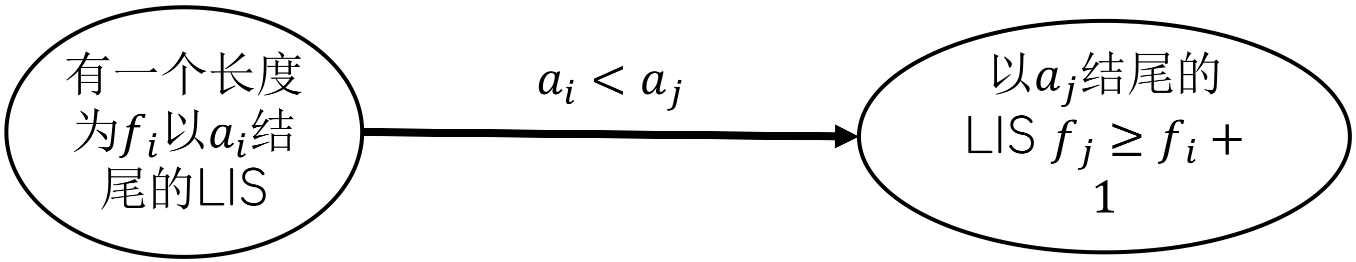
- 如果加入一条边，就把图分成两部分
  - 所以考虑要求加边的时候，图两部分已经连通
- 是否想到了Floyed-Warshell？
  - 接下来是如何避免搜索空间中的重复计算(计数)
  - $f_{i,j,0/1}$表示连通(i,j)，且i是否可以扩展的方案数



总结

DP题千变万化，但思想只有一个

- 找到问题分解的方法 ⇔ 找到搜索空间的合适概括 ⇔ 定义动态规划的“状态”



- 刚开始可能比较难理解
  - 带着搜索空间去理解DP题解
  - 久而久之就对“概括”有感觉了