问题

2019年2月27日 22:22

对于婚姻匹配问题，每一个男的都要求有一个女的相匹配，从而形成配偶
每一个男的都根据对女生的喜欢程度列了一个表（如果可以选择，他们更喜欢和排位高的女生在一起）
每一个女生也根据对不同男生的喜欢程度列了一张表

不稳定的配对：
假设有A（女）和X（男）配对，B（女）和Y（男）配对
对A来说，更喜欢Y而不是X
对Y来说，更喜欢A而不是B
那么A和Y就会出轨，私奔
从而形成更加稳定的配对

要求：寻找一个稳定的配对（即每个情侣都是稳定的，不会出现戴绿帽的情况）

Gale-shapley算法

2019年2月27日 22:28

程序开始时先初始化为全部人都没有配对

每一次都任意抓一个男人出来，直到全部男人都配对完全，程序结束（这意味着全部人都完成配对，即配对过程结束）

- 将这个男人的喜欢列表掏出来，找到剩余可能的配偶中排名最高的一个（因为经过筛选，如果选择恋爱程度最高的那个，必定会被绿）
  - 如果这个女生没有男朋友，就荣誉当选
  - 如果这个女生有男朋友，就把女生的爱慕列表拿出来
    - 如果比他男朋友排名高就可以绿掉他
    - 如果比他低就只能把这个女神从名单中删去

```
Initialize each person to be free.
while (some man is free and hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged, and m' to be free
    else
        w rejects m
}
```

分析

2019年3月2日 15:55

m：男人
w：女人

从算法中可以得出：
1.在求婚的过程中w的伴侣（按照w的偏爱表格）会越变越好
2.在求婚过程中m的伴侣（按照m的偏爱表格）会越变越差
3.while循环最多进行n^2次（即每一个男的选择完他所有的伴侣）

由此可以得出以下推论：

1. 如果m在算法执行过程中是自由的，那么存在一个他还没有向她求过婚的女人
2. 终止时返回的集合S是一个完美匹配（即每个男人都将和一个女人匹配）
3. 考虑GS算法的一次执行，生成的一个小集合S'，这个小集合仍然是一个稳定匹配

证推论一：
假设m是自由而且不存在他没有求过婚的女人，那么所有女人都将与他无缘。
又因为此时必有一个女人单身（男女一一对应），此时他们必能形成一个配对。
然而他们没有形成配对，故与假设矛盾，故推论正确。

推论二证明：
如果有自由的男人，那么这个算法不会终止，当算法终止时必定会有完美匹配

推论三证明：
假设S'没有形成一个稳定匹配，即存在两队配偶(w,m),(w',m')，其中m更喜欢w'，w'更喜欢m，现在分类讨论

1. 如果m之前遇到了w'，那么m明显会选择w'而不是w（m更喜欢w'而不是w），矛盾
2. 如果m之前没遇到w'，那么说明m更不可能遇到w（m肯定先选w'作为自己的对象然后才是w），这个和w与m先相遇并形成配偶矛盾，故舍弃

综上，每一步都是稳定匹配

根据推论三，每一步都是稳定匹配，而有根据推论二，必定会达到终点（即男女一一对应），所以到最后这个算法就是一个稳定匹配算法

推广

2019年3月2日 16:31

对于稳定匹配问题，可能有很多个解
但是通过GS算法得出来的解对于男人来说，
肯定是最优解，即通过GS算法得出来的解对于男人来说一定是最佳有效伴侣
但是对于女人来说，一定是最差的解，即通过GS算法得出来的解对于男人来说一定是最差有效伴侣
通过这个推广，对于同一个稳定匹配问题中，不管GS算法中间的步骤是什么，得出来的结果一定是同样的

最佳有效伴侣：
如果存在一个稳定匹配S包含了（w，m），那么w和m是一个有效伴侣
如果w是m的有效伴侣，且没有w更好的女人是他的有效伴侣，那么就w就是m的最佳有效伴侣

如果将男女的界限拆除，即n个人对（2n - 1）个人都有一个好感列表，那么GS算法就失效

对于男人，如果撒谎，那么他就不会得到最好的结果，但是，对女人来说，如果她们撒谎，那么得到的结果可能是比不撒谎来的好

BFS

2019年3月15日 14:05

BFS，也叫广度优先搜索
含义就是从起点开始，先将根结点的每个子结点遍历一遍（由此可得，队列这个结构就可以实现遍历）
而关于图的bfs，通过这个搜索就会形成一个bfs树（遍历每一个结点的子结点，如果这个子结点出现过，就直接跳过，如果在此之前没有出现过，就把这个结点加到这一层树中）

对于bfs，时间复杂度是O(m + n)，m是边的个数，n是结点个数
对于每一次遍历，都是搜索了自己还有其儿子
总共搜索了n次（总结点数），故相加就是m+n

这个bfs树暗示着从根结点到其他点的最短路径
即可以通过bfs来寻找根结点到其他结点的最短路径（在每一条边权重都相同的情况下）

强连通分量

2019年3月15日 14:44

强连通分量只适用于有向图，即存在一条路径，使得一个集合s中任意两个结点u和v都能相互到达，则称其为强连通分量（这个集合s还要是这个区域内最大的集合）

寻找强连通分量的方法是：对一个结点开始做bfs，判断能否到达其他结点
然后将这个有向图的方向颠倒，在相同的结点做bfs，判断原来能到达的结点是否仍然能到达

DFS

2019年3月15日 14:25

DFS，即深度优先搜索
搜索方式是一条路走到黑，当一条路走不通时再想其他的路
由DFS也可以画出一条DFS树，这条数肯定有一个最长的边。根据这条边上面有若干小分支（对于先遍历的那个路径，每一个结点都是一层，当这条路走不通时，退回来，找到其他没有访问过的子结点，每一个结点的子结点都是用一层表示）

```
DFS(u):
  将 u 标记为"Explored"并且把 u 加到 R
  For 每条邻接到 u 的边(u,v)
     If  v 没有标记为"Explored"then
       递归调用 DFS(v)
     Endif
  Endfor
```

若T是广度优先

用栈进行dfs，每一次循环都要把当前结点pop出来，再加入其子结点，循环n次，故时间复杂度是O(m + n)

通过dfs可以在有向图的判断两个结点u和v是否可达
如果想验证u是否能到达v，则对u为起点做dfs即可
反之同理

二分测试

2019年3月15日 14:35

Bipartite Graphs，也叫二分测试
对于图的每一个结点，只有两种颜色能涂，红和蓝，要求涂同种颜色的结点都不能相邻，问如何涂

解答：对于这个问题，如果这张图是奇图（图上有环，环的结点数是奇数），则不能涂颜色
根据这个性质，想到了BFS，现在只需要要求BFS每一层的结点都不能相连即可。如果有一层有两个结点相连（假设是A和B）那么他们就一定是奇图

一个有趣的现象是：对于每一层的结点，都必须是同一个颜色

图G不是二分图当且仅当G含有奇数边的环

图的基本概念

2019年3月15日 14:19

对于无向图，如果对于图上任意两点u，v都存在一条路径，使得u能到达v，则称这个图是联通的

无向图的环：要求环的结点数量至少为3

有向无圈图

2019年3月15日 14:54

这个又称DAG
可以通过拓扑排序来判断是不是有向无圈图

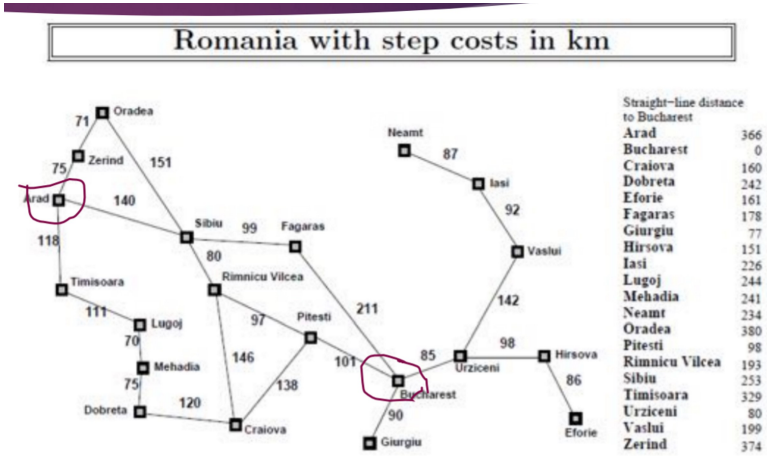
每一个DAG都有拓扑排序（可能有多种排序方式）
而对于有向有环图，一定没有拓扑排序

拓扑排序（这个只针对有向图）：
对于每一个结点，寻找其入度为0的结点，删去该节点，
然后在剩下的结点中判断是否有入度为零的结点，如果有，重复以上步骤直到无结点剩余
如果在操作的过程中无法发现有入度为0的结点，说明这张图有环
在删除的过程中，记录被删除的结点，记录的节点的顺序就是拓扑排序

A*算法

2019年3月15日 15:06

对于A*算法，其基本方式和dijistra算法类似，是dijistra算法的优化版，时间复杂度和dijistra算法类似
解决的问题是：在一个图G中，求a点到b点的最短路径
适用的条件是：我们可以通过别的方式预估各个点到b点的大概路径长度（各个点都在同一个坐标系下）
算法的基本思想是f(x) = g(x) + h(x)，其意思是：
假设现在已经计算到达了x点，f（x）指的是从a点到b点的预估距离（也是我们认为可能的距离），g（x）指的是现在从a点遍历到x点计算的实际距离，h（x）指的是从x点到b点的预估距离



解题步骤是：从A点出发，遍历A点周边的每一个点，由此通过这些子结点就可以算出经过这些子结点所需要的大概路径，且每一次都取最小的预估值进行操作。如果此时停留在的点是B点且此时的预估值最小，算法停止，这个最小的值就是答案
eg：
从A点出发，可以经过的点有：Z点，S点，T点
如果经过Z点，预计的距离是：75 + 374 = 449
如果经过S点，预计的距离是：140 + 253 = 393
如果经过T点，预计的距离是：118 + 329 = 447
所以选择S点
下同

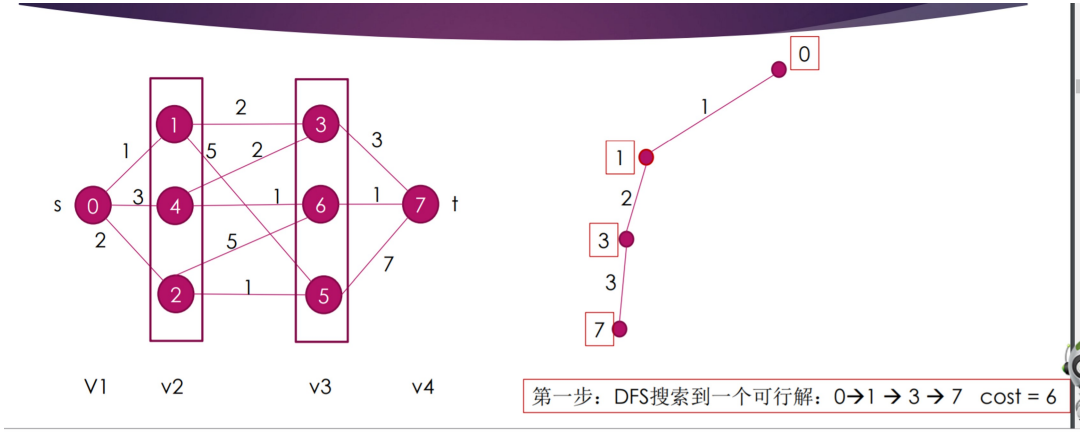
这类算法可以应用于游戏中的自动寻路算法，导航中的指引算法等等

分支界限法

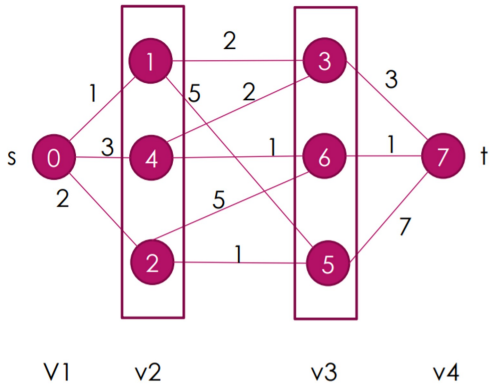
2019年3月15日 15:32

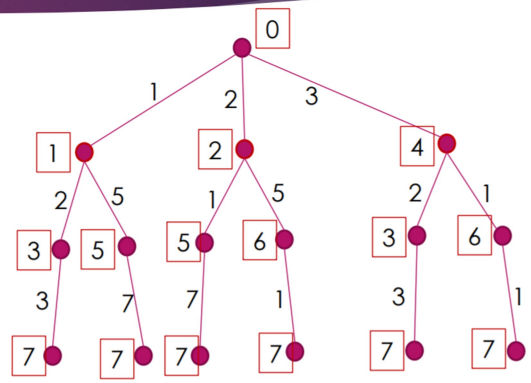
对于分支界限法，是dijistra的另外一种优化，其思想是：先找到一个合适的解，在通过遍历的方式寻找另外一个解，如果新的解在运算过程中就已经比原来的解大，就可以停止运算。如果计算出新的解比原来的小，就需要替换新的解。根据上述步骤类推，直到所有的情况遍历完全。
这种算法的要求是在一开始找到的解法最好就是最优解或者是次优解，这样在剪枝的过程中就可以在寻找最优解的过程中遍历最少的结点

如下图所示，问题是要寻找从s点到t点的最短路径
所有可能的情况已经列出
第一步是，找到一个可行的解
然后对其他可能的情况进行剪枝，直到情况遍历完成

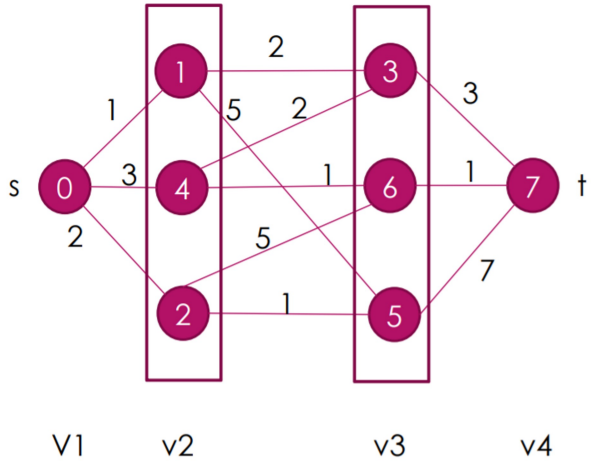


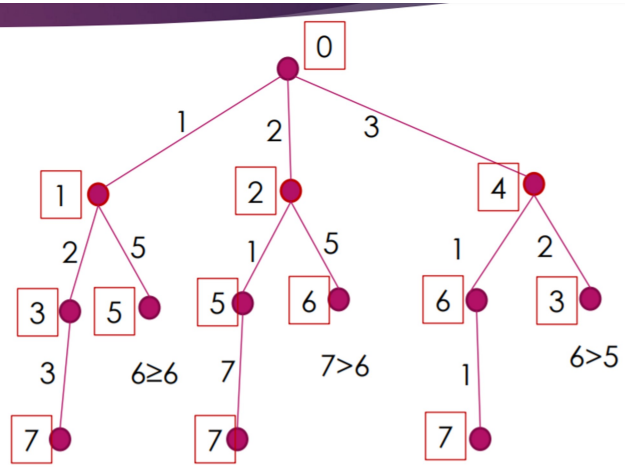
由这个算法可以看出，最坏的情况仍然是遍历完所有的树，优化的关键是事先找到一个次优解，这样算法才能高效的运转，时间复杂度同dijistra算法
在具体的问题下还是要具体的分析





BFS或DFS遍历整棵树，
所有可行路径中找代价最小





越早找到较优的可行解甚至是最优的可行解，剪枝效果越好，算法越高效

区间调度问题

2019年3月23日 13:09

有n项工作，每项工作分别在si开始，ti结束。对每项工作，你都可以选择参加或不参加，但选择了参加某项工作就必须至始至终参加全程参与，即参与工作的时间段不能有重叠(即使开始的时间和结束的时间重叠都不行)。

求在这段时间能参与工作数的最大值

样例:

输入

n=5

s={1,2,4,6,8}

T={3,5,7,9,10}

输出

3(选择工作1, 3, 5)

对于这个问题，使用的是贪心算法，贪心内容是优先选择结束最早的工作来做

```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

  set of jobs selected
A ← φ
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```

Compatible-相容的（工作时间上不冲突的）

区间调度问题证明

2019年3月23日 13:15

假设经过这个算法得到的结果是A，最优解是O

那么只需要证明$|A| = |O|$即可

首先，明显地可以知道：A是一个相容的集合（每个工作之间都不会相互冲突）

假设A得到的结果是m，O得到的结果是n，就是在证明m = n

假设任务i的开始时间是s（i），结束时间是f（i）

A得到的任务序列是a，O得到的任务序列是o

1. 证明：对于任意$0 < i < m$,都有$f(a_i) \le f(o_i)$
   采用数学归纳法，当n = 1时，由贪心策略易得
   当n = k - 1时，可知$f(a_{k-1}) \le f(o_{k-1})$
   当n = k时，由$f(a_{k-1}) \le f(o_{k-1})$，$f(o_{k-1}) \le s(o_k)$
   得$f(a_{k-1}) \le s(o_k)$，不管两个任务是否冲突，都会选时间短的那个，故原命题得证
2. 证明：A是最优解
   也就是证明m = n
   用反证法，假设m < n
   由于$f(a_i) \le f(o_i)$，故$f(o_{m+1})$在$f(o_{m+1})$之后，也在$f(a_m)$之后，且不冲突
   由贪心策略，也会选择$f(o_{m+1})$作为A中的集合，就与原假设矛盾，故A是最优解

由区间调度证明可得，先根据贪心策略证明出一个表达式，然后再根据这个策略来延申

时间复杂度为O(nlogn)

每一个元素都访问一次

每一次都要用堆来求出最优的策略

区间分划问题

2019年3月20日 18:55

有n节课，每一节课的开始时间和结束时间都不一样（这n节课会相互重合）
求如何分配课表使得能利用最小的教室来满足要求

```
Sort intervals by starting time so that s1 ≤ s2 ≤ ... ≤ sn.
d ← 0 ← number of allocated classrooms

for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

首先定义了深度这个概念，深度指的是任一时间都能容纳课程的最大值
根据观察可知：需要的教室数量 >= 深度

贪心的策略是优先取开始时间最小的，如果发现教室数量不够，就增加教室。当一个教室空着，就补充未排布的课程
时间复杂度是O(nlogn)
遍历全部数据，每一次都要寻找最优化的解

证明这个算法是最优解：
就是证明这个算法在这个时间段是最优解
由于要证明最优解，就是证明在d相加时的确是最优解，
也就是这段时间的深度
换句话来说，就是把这个算法复述一遍

最小延迟调度

2019年3月20日 20:16

f(i) 表示某任务 开始的时间。
ti 表示 某任务 加工的时间
di 表示 某任务 要求完成的时间
延迟：f(i)+ti-di
如果 实际完成的时间 小于 规定完成时间，那么，就没有 延迟。延迟就是拖延，如果你在规定时间内（<=规定时间），那么，就说明没有延迟，没有拖延。
由于有很多用户，所以，会有很多不同的拖延值，我们的目标是，求得所有拖延值中的最大值，并使得这个最大的拖延值 最小。Min(Max拖延值)
贪心方法是：
取最短的ddl为先

证明贪心策略：
交换论证
先定义逆序：对于一个调度S，逆序指的是一个工作i和j，i在j的前面被安排，而且di>dj
应该注意到，这个算法产生的结果是没有逆序的
首先，我们知道：存在一个没有空闲时间的最优调度（证明存在性）
其次，我们知道：对于没有逆序的调度都有相同的最大延迟
对于没有逆序的调度，肯定是具有相同的ddl的工作在换来换去，所以都具有相同的最大延迟
而如果减少一个调度就能减少最大延迟，这个是算法证明的关键
关键证明只需要证明两个相邻的逆序即可（不相邻的逆序可以根据多个逆序组合而成，而且相邻的逆序不会导致后面排列顺序的改变）

交换论证
将最优解转换成贪心算法得到的解（最优解可能有多个，贪心算法只是得到了其中一个），就是交换论证
这个论证方法也可以叫构造单调的区间（这里的区间就是逆序）

只有一种情况，就是两个任务都超过了ddl完成（如果没过的话最大延迟就是0了）
只要证明最大的延迟是小于原来的就行了

Optimal Caching

2019年3月27日 19:02

这个讲的是一个收回的调度问题
现在已知任务序列，要求收回相应的调度

算法证明：
先规定”简化“的概念——需要这个任务的时候就收回，其余时间不收回
先证明简化的调度相比于非简化调度所消耗的次数少：
通过列一张表格就行了
然后证明最优解和FF算法是等价的：
通过归纳法，讨论来证明即可，也就是说通过在时间t时刻所做的抉择来证明FF和最优解是等价的

迪杰斯特拉算法

2019年3月29日 15:30

证明这个算法最优：

通过归纳法来证，起始点到任意点的路径都是最短路径

N = 1时，易得

N = k时，假设成立

当n = k+1时，在已知的集合中存在最短的路径

就是需要在已知集合外的某一点到指定点，要证没有最优解（根据迪杰斯特拉算法的性质就可以得出）

注意：此时k到k+1经过了不止一次的循环，而是将所有可能经过v的点都遍历了一遍

最小生成树

2019年4月3日 19:06

通过cut property可以证明prim算法，kruskal

Cut property:通过反证法证明
令S为一张图的子集，令e为S外延的最小边（边权最小），则e必定在S包含的最小生成树内
Cycle property:
对于一个环，其最大的边一定不在最小生成树内

证明Cut property
假设e不在最小生成树T内（假设这个最小生成树为T*），则要形成最小生成树，就需要另外一个边连接两个图的子集，假设这条边为f
由于这两个子集内部互相连接（就是说，除了边e和f，T和T*连接方式一样），而we<wf，所以wT<wT*，所以T*不是最小生成树
故推得矛盾，原假设不成立，cut property得证

对于最小生成树问题，有三种算法：

1. 克鲁斯卡尔
2. Prim
3. 通过降序将各个边通过权值排列，每一次都删除边权最大的边，直到无法形成生成树为止

克鲁斯卡尔和prim算法用的都是Cut property
对于prim算法，就是cut property的直接应用
对于克鲁斯卡尔，每个子集都保证了加到了外部最小的边，所以这样扩展成的就是最小生成树

聚类算法

2019年4月7日 12:42

也就是克鲁斯卡尔
将边添加直到形成k个离散的子集为止
或者先生成最小生成树，然后删除最贵的k-1个边即可

将证明问题转换成和该算法产生的最小的边权d*相比较
通过克鲁斯卡尔的算法步骤判定最小的边肯定不比d*大

哈夫曼编码

2019年4月7日 14:51

首先，对于等长编码，一个很大的缺点是平均二进制位数（ABL）会比非等长编码多，使得编码长度大大增加
为了使得编码长度减小，推出了非等长编码
但是，由于编码是非等长的，所以如果字母A是字母B编码的前缀，就会产生歧义
为了消除歧义，我们引出了前缀码，通过前缀码来消除歧义
通过观察，我们发现使用树的方式进行编码就是前缀码，消除了歧义

为了获得最优的前缀码，我们发现最优前缀码对应的二叉树都是满的
可以通过交换论证来证明：在不影响其他地方编码的影响下改变当前结点，即可完成证明

但是，即使给定了一个树，也不能确定编码的形式。经过观察，有了如下的假设
假设u和v是T*的树叶，使得depth(u)<depth(v)，进一步假设在T*对应于最优前缀码的标记中，
则树叶u的出现频率必定大于树叶v的出现频率
通过交换论证（如果u和v交换，会怎么样）和公式推导来比较两者的大小

通过观察和上述推到可以得出哈夫曼的算法
就是每一步将最低频率的两个字母构成一个树，并删除这两个字母，构造出一个新的字母w，将新的字母加到树中，直到字母表只剩下一个数字

证明哈夫曼树时编码最优
根据题意，可知，在将旧字母y和z替换成w时，假设原本的树为T，替换后的树为T*，则$ABL(T) = ABL(T*) + f_w$等式通过w的深度比y和z少一，还有$f_w = f_y + f_z$得到
故可根据反证法，证明编码最优
假设存在另外一种编码树，使得ABL最小，设为Z，即ABL（Z）<ABL（T）
则ABL(Z*)<ABL(T*)（根据上述等式得到），而T*是最优的，（T*经过无数次迭代，变成只有两个叶子的树，只有两个叶子的树明显是最优的），故不存在Z*，使得其比T*更优，故原假设矛盾

判断逆序对个数

2019年4月17日 16:51

对于逆序对的判断，使用归并排序
在排序的过程中找到点对即可，如果发现有逆序，对应相加即可

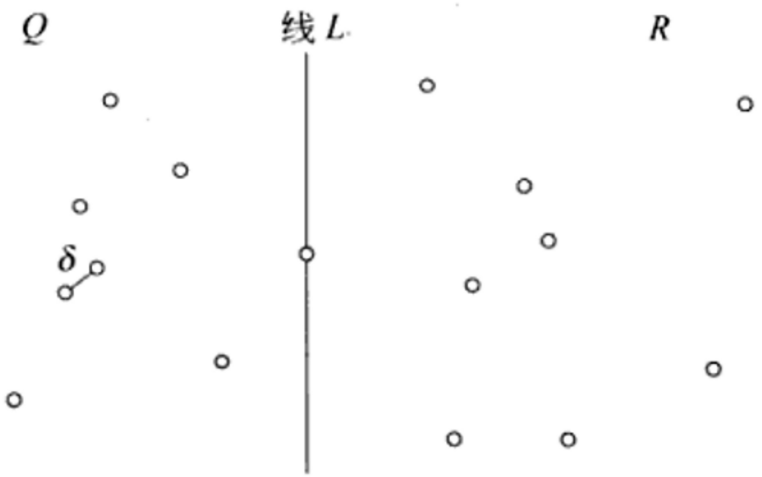
寻找最近的点对

2019年4月17日 17:28

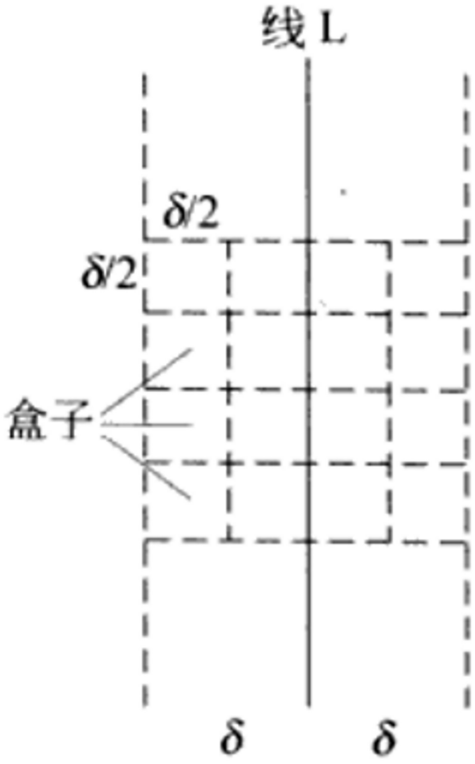
首先，将这些点分别按照x轴和y轴进行排序
通过x轴的前后一半的点，可以将这些点集分成两部分
通过递归找到这两部分的局部最小解，然后将其合并成全局最优解即可

时间复杂度分析
$T(n) = 2(T/2) + O(n)$
所以说时间复杂度为$O(n\log n)$

假设局部最优解为σ



由于我们已经知道了Q和R上的最优解，则合并的最优解是点一边在Q，一边在R上
故只需找到离中间线水平距离相隔为σ的点，也就是说，只需要找到在条带上面的点即可
假设对左边的点进行遍历，由于已经排好序，则只需寻找条带右侧上下距离为σ的点即可


线 L
δ/2
δ/2
盒子
δ
δ

归并排序及其优化

2019年4月18日 13:31

关于归并排序的算法，伪代码如下：

```
输入：序列L
输出：排好序的序列L以及逆序对的数目r

Sort-and-Count(L) {
    if list L has one element
        return 0 and the list L
    Divide the list L into two halves A and B
    //递归调用，返回A中的逆序对数目rA及sorted A
    (rA, A) <= Sort-and-Count(A)
    //递归调用，返回B中的逆序对数目rB及sorted B
    (rB, B) <= Sort-and-Count(B)
    //归并排序，输入sorted A及sorted B，
    //得到sorted L，排序的同时统计逆序对r
    (r, L) <= Merge-and-Count(A, B)
    return (r=rA +rB +r, the sorted L)
}
```

```
输入：排好序的A及排好序的B
输出：排好序的序列L以及逆序对的数目r
Merge-and-Count(A, B){
    创建序列L，长度为AB的长度之和
    r初始化为0
    i指向A的首元素，j指向B的首元素
    while  i没有超过A的末尾并且 j没有超过B的末尾
        if (A[i] <= B[j])
            将A[i]写入L
            i后移
        else
            将B[j]写入L
            j后移
            r +=A中i以及i之后剩余元素的数目
        endif
    endwhile
    如果A中i之后还有元素，将剩余的元素都依序写入L
    如果B中j之后还有元素，将剩余的元素都依序写入L
    return (r, the sorted L)
}
```

merge时，每一次都要开一个新的数组，很浪费时间

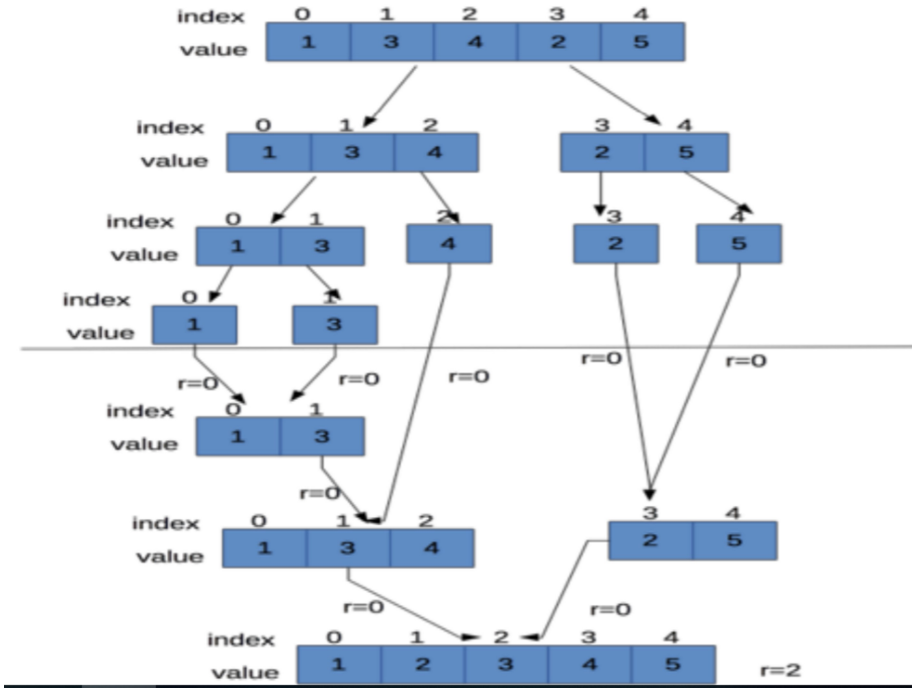
由此想到：直接创建一个大小为n的数组，来作为辅助数组，并且在拷贝的时候通过原始数组来拷贝即可

merge的过程中，每一次实现外排过后都要将辅助数组复制一次，遍历过程十分麻烦

解决方法：在使用递归时将真实数组和辅助数组调换位置，使得本轮永远是将真实值外排到辅助数组中（这样上一轮正确排序的数组就变成了真实数组）

首先，根据递归树，发现在递归的过程中，如果右边第一个数字大于左边的最后一个数字，就不用外排，直接拷贝即可。

由此想到：先进行一次比较，来判断是否大于。如果大于，直接拷贝，否则进行外排



对于小数据量的数组运行时间很慢

解决方法：当数据量小到一定程度时，转换成别的排序算法

修改过后的源代码：

```
auxL = L.clone
Sort-and-Count(L,auxL,low,high)
   如果只有一个元素，返回0
   mid = lo + (hi - lo) / 2
   // low到mid为A段，将auxL的A段排好序放到L的A段
   rA = Sort-and-Count(auxL, L,low,mid)
   // mid+1到high段为B段，将auxL的B段排好序放到L的B段
   rB = Sort-and-Count(auxL, L,mid+1,high)

   if (L的A段的最后一个元素小于等于B段的第一个元素)
     将L的AB拷贝到auxL的AB段
   else
      //将上述L的AB段进行merge放到auxL段
      r = Merge-and-Count(L,auxL,low,mid,high)
return (rA+rB+r)
```

对于merge的操作，则省去了拷贝数组的过程

整数乘法

2019年4月18日 17:59

在计算机的整数乘法中，通常是二进制计算
方法和小学学到的方法相同，就是将n个数错位相加
对于一个数的加法时间复杂度为$O(N)$
所以乘法就是$O(N^2)$，于是想到用分治来优化算法

假设有两个大小为n（占用bit数目）的两个数a,b相乘
将ab化简成$ab = (x_1y_1)\cdot 2^n + ((x_1+x_0)\cdot(y_1+y_0) - x_1y_1 - x_0y_0)\cdot 2^{n/2} + x_0y_0$
也就是遵从表达式$T(n) = 3T(n/2) + O(n)$
从而得到了化简

快速傅里叶变换

2019年4月18日 18:42

问题

令$A(x)=a_0+a_1x+\ldots+a_{n-1}x^{n-1}$，$B(x)=b_0+b_1x+\ldots+b_{n-1}x^{n-1}$，$C(x)=A(x)\cdot B(x)$（$a_0,a_1$，$b_0,b_1$等系数已知），求C（x）的系数

暴力解

只需套用公式$c_k=\sum_{i+j=k}a_i\cdot b_j$，k = 0,1...2k-2

此过程需要$O(n^2)$次乘法，总体时间复杂度$O(n^2)$

解决多项式的额外方法

可以将暴力计算系数的方式转换成计算带点计算的方式

如果确定n个不同的点，计算出$A(x_i),B(x_i)$，则$C(x_i)$就可以通过AB相乘得到结果

由于n个等式可以确定n个方程，故只需要带进去n个点即可算出对应的系数

但是，如果带了n个不同的点，则每一个点的计算量就很大

所以，要想时间复杂度足够小，就要带足够规律的点

于是，想到了复平面上的定理：在复平面上，$x^n=1$有n个不同的根，而且这n个根相互关联，从而简化了计算

于是，带点的时候就只需要带这些点就可以大大简化计算量

但是，这样的时间复杂度还是$O(n^2)$

于是想到了分治算法

按A(x)下标的奇偶性把A(x)分成两半，右边再提一个x

$A(x) = (a_0 + a_2x^2 + \ldots + a_{n-2}x^{n-2}) + x(a_1 + a_3 + \ldots + a_{n-1}x^{n-2})$

于是可以得出

$A(x) = A_1(x^2) + xA_2(x^2)$

再设$k < \frac{n}{2}$，把ω_n^k作为x代入$A(x)$~~(接下来几步变换要多想想单位根的性质)~~

$$A(\omega_n^k) = A_1((\omega_n^k)^2) + \omega_n^k A_2((\omega_n^k)^2)$$

$$= A_1(\omega_n^{2k}) + \omega_n^k A_2(\omega_n^{2k}) = A_1(\omega_{\frac{n}{2}}^k) + \omega_n^k A_2(\omega_{\frac{n}{2}}^k)$$

那么对于$A(\omega_n^{k+\frac{n}{2}})$的话，代入$\omega_n^{k+\frac{n}{2}}$

$$A(\omega_n^{k+\frac{n}{2}}) = A_1(\omega_n^{2k+n}) + \omega_n^{k+\frac{n}{2}} A_2(\omega_n^{2k+n})$$

$$= A_1(\omega_n^{2k}\omega_n^n) - \omega_n^k A_2(\omega_n^{2k}\omega_n^n)$$

$$= A_1(\omega_n^{2k}) - \omega_n^k A_2(\omega_n^{2k}) = A_1(\omega_{\frac{n}{2}}^k) - \omega_n^k A_2(\omega_{\frac{n}{2}}^k)$$

于是只要求出$A_1(\omega_{\frac{n}{2}}^k)$和$A_2(\omega_{\frac{n}{2}}^k)$即可知道$A(\omega_n^k)$和$A(\omega_n^{k+\frac{n}{2}})$

由此根据将多项式分成奇偶两种形式来体现分治思想

对于B也是同理，最终时间复杂度为$O(n\log n)$

带权重的区间调度问题

2019年4月28日 18:16

简单来说，就是区间调度问题加上权重
要求得到的结果权重最大
原本这是一个贪心的问题，加上权重后变成了动态规划问题

首先，按照结束时间将各个任务从小到大排序
定义p(j) = largest index i < j such that job i is compatible with j.
则可以列出以下等式

$$OPT(j) = \begin{cases} 0, if\ j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\}, otherwise \end{cases}$$

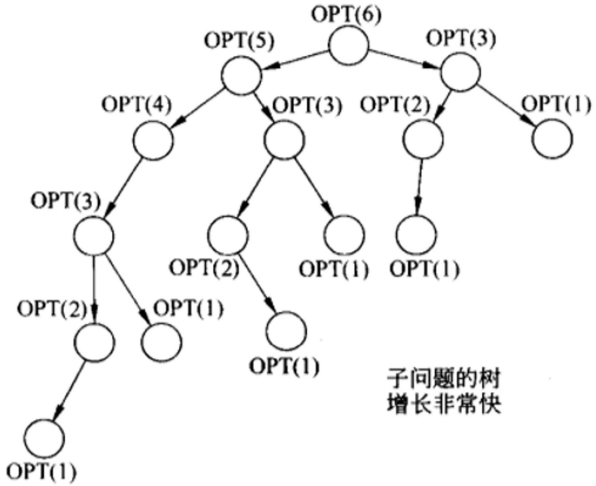
根据这个等式，就可以通过递归得到最终的结果（只需令j为最大的那个任务即可）

可以使用归纳法来证明这个状态方程的正确性
由于证明很容易，这里省略

不过这样的方式会计算出不需要的数据
所以，通过递归（反向追踪）的方式合理的计算子问题

```
Find-Solution(j)
    If j=0 then
        什么也不输出
    Else
        If v_j + M[p(j)] ≥ M[j-1] then
```

如果直接使用递归，就会变成指数的时间
原因在于：计算OPT（p（j））时，会有很多重复的数字被计算



所以优化的方法是：
创建一个数组，通过迭代的方式将1到i这些问题算出来
使用bottom-up的方式来求解

```
Find-Solution(j)
    If j=0 then
      什么也不输出
    Else
      If v_j + M[p(j)] ≥ M[j-1] then
        输出 j 与 Find-Solution(p(j))的结果
      Else
        输出 Find-Solution(j-1)的结果
      Endif
    Endif
```

分段最小二乘

2019年5月16日 0:49

问题描述：
对于二维平面上的点，使用最小二乘法来进行线性拟合
拟合的直线为：y=ax+b
公式为：

$$\mathrm{Error}(L,P)=\sum_{i=1}^{n}(y_i-ax_i-b)^2$$

$$a=\frac{n\sum_i x_iy_i-(\sum_i x_i)(\sum_i y_i)}{n\sum_i x_i^2-(\sum_i x_i)^2} \quad \text{和} \quad b=\frac{\sum_i y_i-a\sum_i x_i}{n}$$

现在，给定一组点对P，要求分成若干段，每一段用直线表示都是尽量的线性拟合
也就是说，假设这个点对被分成了p份，变为OPT，现在要求的是OPT

时间复杂度$O(n^3)$
对于使用i和j遍历数据用了$O(n^2)$的时间复杂度
每一次计算最小二乘误差需要$O(n)$的时间复杂度
所以，总共需要$O(n^3)$的时间复杂度

根据观察，我们得到

定理 6.6 如果最优划分的最后一段是

$$\mathrm{OPT}(n)=e_{i,n}+$$

定理 6.7 对于点 $p_1,\cdots,p_j$ 上的子问题，

$$\mathrm{OPT}(j)=\min_{1\leqslant i\leqslant j}(e_{i,j}+$$

并且段 $p_i,\cdots,p_j$ 被用于一个关于子问题的最优解

所以得到的伪代码如下：

```
Segmentd-Least-Squares(n)
  数组 M[0,…,n]
  置 M[0]=0
  For 所有的对 i≤j
    计算对于段 p_i,…,p_j 的最小二乘误差 e_{i,j}
  Endfor
  For j=1,2,…,n
    使用递推式 6.7 计算 M[j]
  Endfor
  返回 M[n]
```

$p_i, \cdots, p_n$，那么最优解的值是：

$C + OPT(i-1)$.

$C + OPT(i-1))$

，当且仅当这个最小是使用下标 i 得到的.

01背包问题及其扩展

2019年5月16日 16:12

假设有n件物品，第i件物品价值是v_i，重量是w_i

现在有一个载重为W的背包

问：应该如何安排放入的物品，使得放入的物品价值最高

这类问题又叫子集和问题

这道题不适用于任何贪心的策略（不管采用什么贪心策略都会找到一个反例）

只能用动态规划求解

解法的思路是自顶而下的

当遍历到第i件物品时，假设此时背包的重量是w，最优解的价值为$OPT(i,w)$

对于第i件物品，只有两种可能：选，或者不选

如果选，式子就是$OPT(i,w) = v_i + OPT(i-1, w-w_i)$

如果不选，式子就是$OPT(i,w) = OPT(i-1,w)$

由于是最优解，所以要有取两者的最大值

$OPT(i,w) = \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\}$

由于有W的限制，所以当选择后的重量 < 0，就只能不选

$OPT(i,w) = OPT(i-1,w)$

而且，还需要一个初始值

$OPT(0,W) = 0$

所以得到了状态方程

$$OPT(i,w) = \begin{cases} 0,\ i=0 \\ OPT(i-1,w),\ w < w_i \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\}, else \end{cases}$$

由于是两个变量i和w，所以最终的表是一个二维数组

完全背包问题：

对01背包问题做了扩展，每种物品可以放无限多个

则这个问题就有很多个可选，即放置一个物品，或者放入多个，或者不放

$$OPT(i,w) = \begin{cases} 0,\ i=0 \\ OPT(i-1,w),\ w < w_i \\ \max\{OPT(i-1,w), OPT(i-1,w-kw_i)(1 \le k \le \left\lfloor \frac{w}{w_i} \right\rfloor)\}, else \end{cases}$$

或者可以写成如下等式

$$OPT(i,w) = \begin{cases} 0,\ i=0 \\ OPT(i-1,w),\ w < w_i \\ \max\{OPT(i-1,w), OPT(i,w-w_i)\}, else \end{cases}$$

下面的状态方程指的是每一次选择，都是选这一件物品，或者不选，如果选择了这一件物品，可以通过下一次迭代来再选改件物品，直到剩余的重量不够为止（就走第二个条件）

多重背包问题

在01背包问题的基础上添加了一个新条件：每种物品有一个固定的次数上限

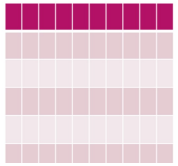
由于物品的总数还是有限的，所以完全可以将每种物品的每一件商品看成是独立的个体，而不是通过数量来表示

weight = 1500

物品种类	value	weight
0	13	5
1	17	7
2	22	8

编号	种类	value	weight
1	0	13	5
..	0	13	5
300	0	13	5
301	1	17	7
..	1	17	7

702*1501



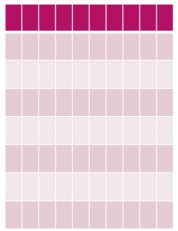
由于是两个变量i和w，所以最终的表是一个二维数组

weight = 1500

物品种类	value	weight
0	13	5
1	17	7
2	22	8

编号	种类	value	weight
1	0	13	5
..	0	13	5
300	0	13	5
301	1	17	7
..	1	17	7
514	1	17	7
515	2	22	8
...	2	22	8
701	2	22	8

702*1501



但是这样又会有新的问题，如果商品的数量太多，会导致空间占用很大而且很耗时间

根据相同的种类商品的参数完全相同的特点，可以将相同种类的商品转换成参数不同的个体

对于一个种类的商品，关键在于数量上的选择

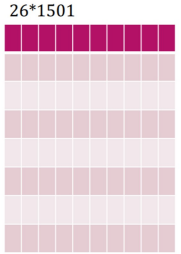
这里的方法是：对于上面的例子，将种类0的数量300转换成二进制，并赋予到每一件商品中

weight = 1500

物品种类	value	weight
0	13	5
1	17	7
2	22	8

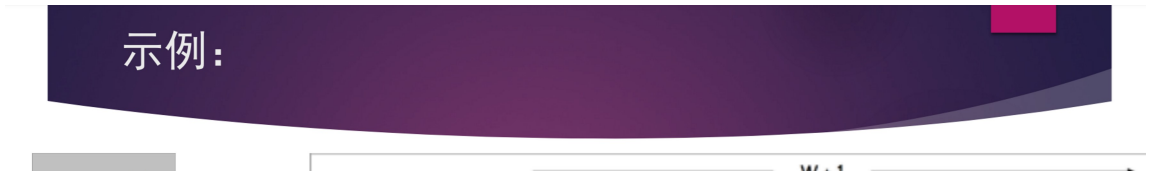
编号	种类	value	weight
1	0	13*2^0	5*2^0
...	...	...	...
9	0	13*2^8	5*2^8
10	1	17*2^0	7*2^0
...	...	...	...
17	1	17*2^7	7*2^7
18	2	22*2^0	8*2^0
...	...	...	...
25	2	22*2^7	8*2^7

26*1501



通过这种方式，也可以唯一的确定相同种类的商品的数量，而且时间空间的复杂度更小

下面是w=11时的一个例子，w指的是现在最多情况下的载重，ϕ指的是总物品组成的子集



二维背包问题

在01背包的基础上增加了一个条件，物品体积的限制（用V表示）

W = 11

#	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W + 1

n + 1

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0	1	1	1	1	1	1	1	1	1	1	1
{1, 2}	0	1	6	7	7	7	7	7	7	7	7	7
{1, 2, 3}	0	1	6	7	7	18	19	24	25	25	25	25
{1, 2, 3, 4}	0	1	6	7	7	18	22	24	28	29	29	40
{1, 2, 3, 4, 5}	0	1	6	7	7	18	22	28	29	34	34	40

根据其状态方程

$$OPT(i,w)=\begin{cases}0,\ i=0\\ OPT(i-1,w),\ w<w_i\\ \max\{OPT(i-1,w),v_i+OPT(i-1,w-w_i)\},else\end{cases}$$

当前状态只和原来i-1的状态有关，而和之前的状态无关，所以可以对算法进行空间上的优化，即：只设置长度为W * 2的数组，分别保存$OPT(i,w)$和$OPT(i-1,w)$的情况，当算出$OPT(i,w)$时立刻复用原来的数组，节省了空间

对01背包问题再多一个条件，要求物品刚好装满背包

由于要刚好装满背包，所以在初始状态下需要恰好装满背包

即当ϕ=0时，点W = 1，2，。。。。n全部要设为负无穷

就是说这样子不符合条件

通过这样的初始化，达到了所要求的题意

二维背包问题

在01背包的基础上增加了一个条件，物品体积的限制（用V表示）

解就是使用三个参数来表示（是否选用该物品，背包的重量，背包的体积）

原本的二维数组也变成了一个三维数组

从而得到了解答

Bellman-Ford 算法

2019年5月31日 20:14

当图中出现负值时就不能用dijkstra算法了

Bellman-Ford 采用的是动态规划的思想，状态方程为

$OPT(i,v) = \min\{OPT(i-1,v), \min_{w \in V}\{OPT(i-1,w) + c_w\}\}$

其中，i指的是从起点到v需要i步，v指的是目前遍历的点

Bellman-Ford采用的算法是遍历所有的点

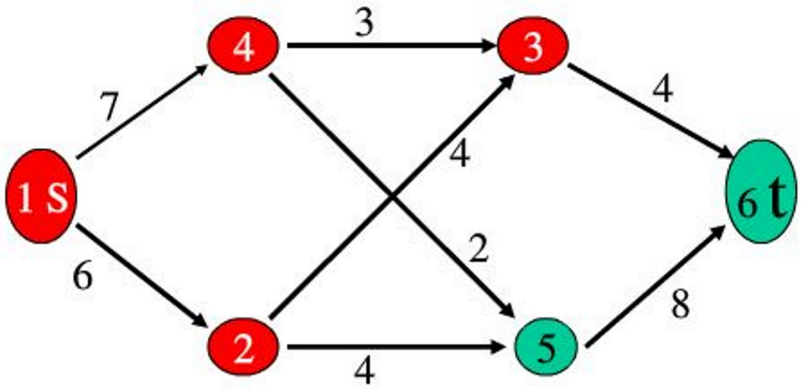
dijkstra采用的是贪心的策略，并只是在一个特定的边中取了所有的点

问题描述

2019年5月31日 19:09

实例：

有一自来水管道输送系统，起点是S，目标是T，途中经过的管道都有一个最大的容量。



- 问题：问从S到T的最大水流量是多少？

最大流问题

2019年5月16日 18:08

在一个连通图中，如果删掉若干条边，使图不联通，则称这些边为此图的一个割集。在这些割集中流量和最小的一个称为最小割。
最大流最小割定理：一个图的最大流等于最小割。

其中，最大流问题要求的是满足物质守恒定律（一个结点流入的流量等于流出的流量）
而且，不会凭空产生其他流量，只可能从一个地方产生流量

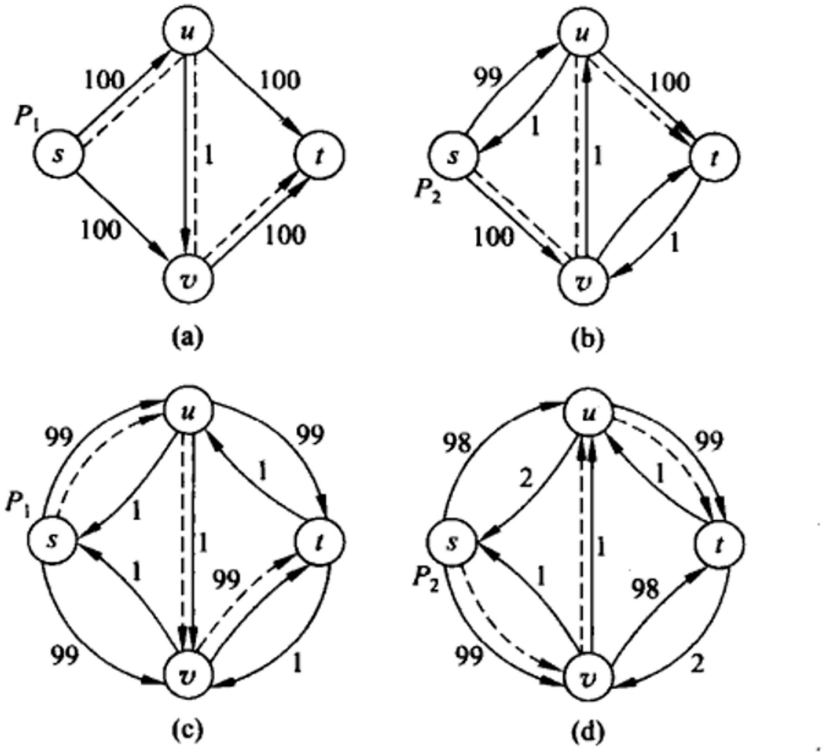
对于s（起点）来说，保证s只有出度的边，t（终点）只有入度的边
要求找到结点的集合（包括s）使得这个超节点的出度边权值最小

FF算法

2019年5月31日 19:13

先找到一个增广路径（满足题意的流的路径）
其流量的值为$\min\{(u_i, v_j), i, j = 1,2\ldots\}$
将这个路径的对应值反向，再找一个增广路径，直到图中不再存在增广路径

但是，这样的算法在遇到一些例子时会很低效



对于这个例子，FF算法就要执行200次

FF算法优化

2019年5月31日 19:26

EK算法：
又称最短增广路算法，指的是每一次找增广路径时都找最短的路径
这个通过BFS来寻找增广路

优化方法是：
设置一个限制参数Δ
要求一开始增广路径时流量至少是Δ
如果没有找到，就将Δ变为原来一半
这样通过2的倍数来进行限制的Δ，就可以提高效率

Dinic算法：
当处理完一个增广路径后，EK算法要重新回到起点开始找，这就很麻烦
Dinic算法对这个进行了进一步优化
首先，对图进行一个bfs处理，找出同层次的点
寻找流的过程中只找相邻的点，而不是同层次的点（遍历到想同层次上的点时，只是在对流量进行储存，而没有流向汇点）
寻找增广路径时，使用dfs
当找到增广路径时，对路径进行回溯，从而减少了时间复杂度

稳定匹配

2019年5月31日 13:43

男生的配偶只会越来越差
女生的配偶会越来越好
总能保证男女匹配问题有解

图

2019年5月31日 13:46

寻找连通分量：

BFS

判断一张图是否为二分图（任何一条边都有红蓝两个节点）：

BFS，画出BFS树，每一层之间不能有相连的边

一张图不是二分图当且仅当有边长为奇数的环

判断一张图是否为强连通图：

两次BFS，正面一次BFS，在反向跑一次BFS

DAG——有向无环图，判断一张图是否为DAG：

即判断是否有拓扑序。找拓扑序方法：

a. 从起始点开始 一路DFS 出栈时加入列表 最后反向输出
b. 每次删入度为0的点删除的点放入列表

贪心总结

2019年5月31日 13:41

证明贪心的方法：

交换论证

1. 设原贪心策略得到的结果是A，最优解为O，A不是最优解，A和O在前r-1个元素上相同，在第r个元素开始出现不同
2. 修改O的算法，变成O’，证明O’也是一个最优解
3. 通过重复以上步骤得到A也是最优解（到最后O’和A相同）

变种：

1. 给出贪心算法A
2. 假设最优解是O
3. 找出A和O的不同点（不同点可能在于元素不同，或者顺序不同）
4. 交换其中一个不同的元素，使得O变成O’，证明O’也是最优算法
5. 证明同类型的元素还有有限个，只要把这些东西都交换掉，采用同样的方法，就可以将O变成A
6. A是一个最优解

对于交换论证，有时先证明一个充分条件来限制贪心的策略

通过观察，给最优解一个界限：

1. 假设界限为A，即最优解肯定大于等于A
2. 分析给出的贪心算法，对于算法执行的每一步都证明严格等于A
3. 该贪心算法为最优解

割性质：

环性质：

网络流问题

2019年5月31日 15:25