搜索

主讲人：林哲涵

*什么是搜索

搜索主要分为深度优先搜索(dfs)和广度优先搜索(bfs)两种，可以用于图的遍历或方案的枚举，实质上都是状态空间的遍历，在需要考虑较多状态时可以考虑搜索

由于要枚举大量状态空间，所以搜索一般复杂度较高，因此优化搜索方式，及时排除无效搜索是一门高深的学问，如果能设计出好的搜索，在各类比赛中虽不一定能满分，但还是能获得不错的分数。

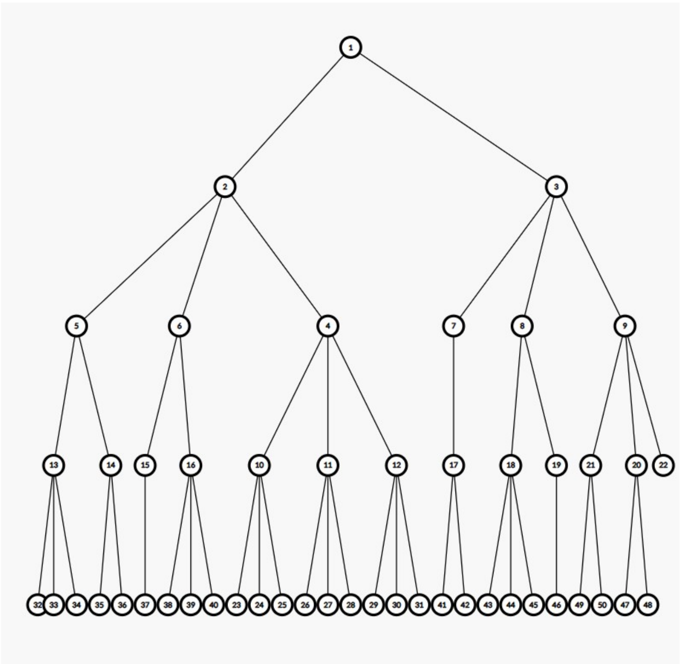
搜索为了保证效率，不会重复访问同样的状态，所以最终遍历的路径形成一棵树形结构，称为搜索树

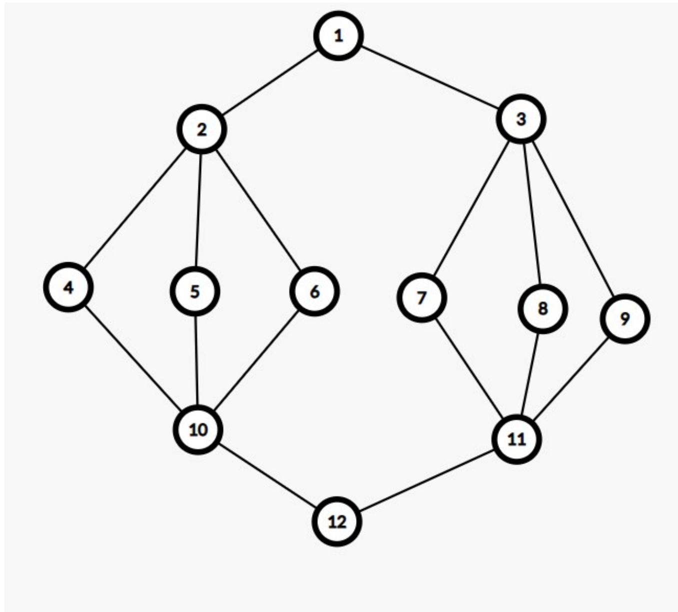
剪枝

剪枝是优化搜索的常见技巧之一，比较具有灵活性与综合性，需要据题意而定。剪枝虽不能优化理论复杂度，但可以排除许多无效、冗余的搜索，使实际复杂度远远达不到理论复杂度，至少能降一个数量级。我们会在后面的具体题目中讨论剪枝的方法。

双向搜索

搜索树的节点数随着层数的增加，可能会有指数级的增长，所以减少搜索次数的一个重要思想就是减少搜索层数。有时，问题既有初态，又有明显的终态，就可以从起点和终点开始各搜一半的深度，最终在中间交汇，就可以大大提高搜索效率。





例题：P3067 Balanced Cow Subsets G

- 定义一个集合S是平衡的，当且仅当满足以下两个条件:
- - S非空。
- - S可以被划分成两个集合A,B，满足A里的元素之和等于B里的元素之和。划分的含义是，$A \cap B$为$\varnothing$且$A \cup B$为S。
- 现在给定大小为n的集合S，询问它有多少个子集是平衡的。请注意，数字之间是互不相同的，但是它们的值可能相同。
- 对于全部数据，$1 \leqq n \leqq 20, 1 \leqq a_i \leqq 10^8$

对于每个数，有三种选择：作为第一部分，作为第二部分，不作为任何部分。枚举每个数的状态，同时记录选择方案，如果第一部分之和等于第二部分之和且选择的方案之前没贡献过，将答案加1，方案标记为已贡献。复杂度$O(3^n)$，无法接受。

我们发现$O(3^{n/2})$是能过的，所以考虑折半，把数列分成前一半和后一半。设前一半作为第一部分的数和为a,作为第二部分的数和为b,后一半作为第一部分的数和为c,作为第二部分的数和为d,显然当a+c==b+d时会产生贡献，变形得a-b==d-c。这样，我们在搜索第一部分时可以记录第一，二部分数的差，同时记录选取方案，搜完后对差离散化，把方案放在vector对应的差里。搜第二部分时，记录第二，一部分数的差和选取方案，搜完后在对应的vector里检查，如果方案还未贡献过就把答案加一。复杂度$O(3^{n/2})$。

- CF525E Anya and Cubes
- 给你n个数,初始序列为a
- 你有k个!,每个!可以使序列中的一个数变成a_i!，每个数至多用一次!
- 求:对于每种可能的操作后，选出任意个数,使他们和的等于S的方案数之和。
- $1 \leqq k \leqq n \leqq 25, 1 \leqq a_i, S \leqq 10^{16}$

对于每个数，有三种选择：不选，选a_i,选a_i!。一个明显的剪枝是若$a_i \geq 19$就不使用！。复杂度$O(3^n)$，无法接受。

考虑折半，搜索时记录当前数的和，还剩多少个！。搜出的结果用一个map<pair<int,long long>,int>,记录前半部分用了i个！，和为j的方案有多少。搜后半部分时，若后半部分用了h个！,和为e，就把所有满足j==s-e且i+h<=k的map值贡献到ans上。复杂度$O(3^{n/2}+k*3^{n/2})$。

由于后半段复杂度多了个k,所以我们可以让后半段少搜一点，让前半段多搜一点，让复杂度平均，如把“前半段”定义为1~n/2+2,后半段定义为n/2+3~n,这样实际效率会更高。

CF912E Prime Gift

- 给你n个互不相同的素数p_1,p_2……它们组成一个集合P。请你求出第k小的正整数，满足：该数字的所有素因子∈P。
- 1≦n≦16，2≦p_i≦97，保证答案不超过10^{18}。时限3.5s。

- 我们可以二分答案，通过判定有多少个不超过mid的满足条件的数调整二分上下界。我们有一个直接的想法，就是暴力搜出所有小于1e18且满足条件的数后去重，排序，这样就能再二分找出有多少满足不超过mid的数。
- 在1e18范围内满足条件的数至少是1e12级别的，时间和空间都无法承受。那我们又可以考虑折半，考虑有只用前一半质数和只用后一半质数能组成哪些数。如果只考虑8个质数，那搜出来的满足条件的数是5e6级别的，可以承受。把两次搜索搜出来的数排序去重，存在数组A和B内。
- 当询问有多少个不超过mid的满足条件的数时，我们从前往后遍历数组A，记录个指针j,初始指向B的末尾。对每个A_i，我们让指针往前扫，找到第一个使$A_i*B_j \leq mid$的j，答案就加j。
- 就做完了。复杂度最高为$O(5e6(\log_2 5e6+\log_2 1e18))$,最慢的点跑了950ms。
- 注意加上一个剪枝：把p排序，第一次搜排名为奇数的质数，第二次搜排名为偶数的质数，这样可以平均搜索数量，减小总复杂度。

迭代加深搜索

由于搜索树的节点数随着层数的增加，会有指数级的增长，所以普通的dfs很容易由于进入错误的子树，而在上面浪费许多时间，而往往答案只在较浅的节点上。因此，我们可以考虑限制搜索深度，如果搜索深度大于限制就返回无解，再加大限制，直到找到解。这样的话，我们就只会在浅层搜索树上废时间，而不会进入深层的子树。

- P1763 埃及分数

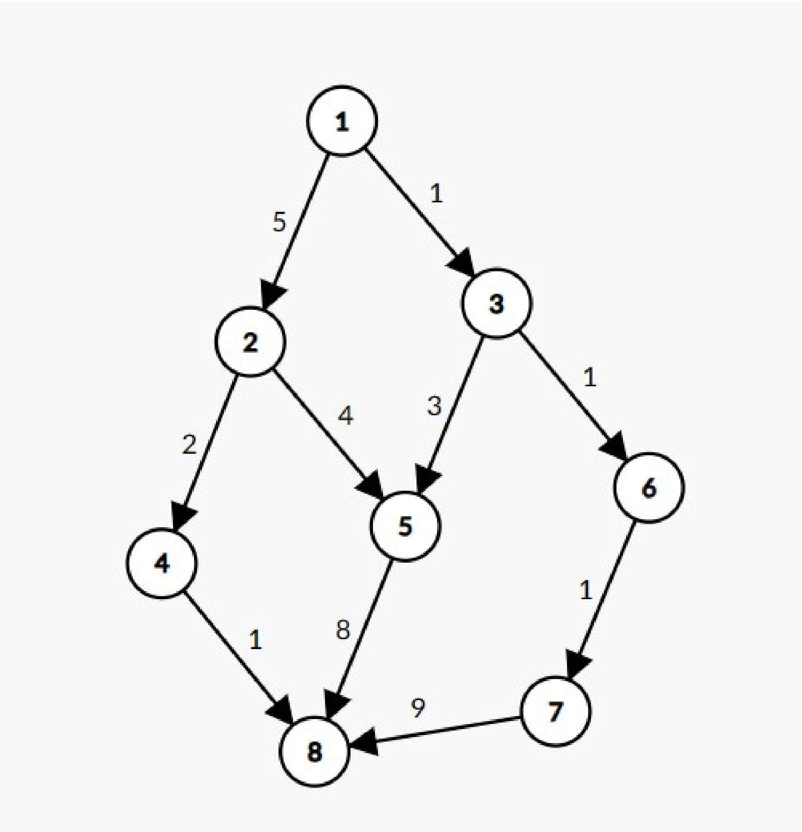
[P1763 埃及分数 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)](luogu.com.cn)

- 题目要求在数字最少的前提下最大分母最小，那我们在迭代加深搜索是可以做两个限制，一个是个数限制，一个是最大分母范围限制，在不超过范围的情况下搜解，超过限制就返回无解。之所以要限制最大分母的大小，还有一个原因是因为这有利于剪枝。为了保证效率，我们把分母上限ax初始设为10000，每次*5，直到1e7；你可以用你喜欢的方式记录方案，我是搜了两遍。
- 这样就能100pts了，但过不了，因为有hack数据。我们可以加入几个高妙的剪枝：

- 设接下来选的分数为 $\frac{1}{i}$ ，当前考虑分数$\frac{a}{b}$,单位分数个数限制为cnt，最大分母限制为ax,正在枚举第now个分数;
- 0、从小到大枚举分母。记录上次的选择，这次就从那开始枚举
- 1、首先必须满足$\frac{a}{b} \geqq \frac{1}{i}$,及i$\geqq \frac{b}{a}$;
- 2、因为后面选的分数小于当前分数，所以必须满足 (cnt−now+1)* $\frac{1}{i}$ >= $\frac{a}{b}$,及i<=(cnt-now+1)* $\frac{b}{a}$;
- 所以，我们第now个分母枚举的范围就缩小了许多
- 若now==cnt,则直接考虑$\frac{a}{b}$是否合法，无需考虑下一层
- 若now==cnt-1,则还剩两个分数,设$\frac{a}{b}$= $\frac{1}{x}$+ $\frac{1}{y}$ $= \frac{x+y}{xy}$,及a,b同时乘上个z,会变成x+y和xy,可得到方程组$\begin{cases} az = x + y \\ bz = xy \end{cases}$。因此可枚举z,解出x,y,判断是否合法。自己手算一下有解条件即可算出z的上下界。
- 不要小看最后两个剪枝，它们剪掉了最后一两层的搜索，而搜索的规模是指数级的，所以还是很有用的。
- 使用以上剪枝后就足以通过本题，还有其它更奥妙的剪枝,可以参考第一篇题解

启发式搜索

- 基础的启发式搜索主要包括A*和IDA*,分别是对bfs和dfs的优化。如果给定一个目标状态，求初状态到目标状态的最小代价，我们就可以考虑A*和IDA*。我们考虑用dijkstra求下图的最短路：



队列:id	dis	取出	插
1	0	1	2,
3,2	1,5	3	5
6,5,2	2,4,5	6	7
7,5,2	3,4,5	7	8
5,2,8	4,5,12	5	8
2,8,8	5,12,12	2	4
4,8,8	7,12,12	4	8
8,8,8	8,12,12	8(得出答案)	

我们发现，真正的最短路1-2-4-8到很晚才得以扩展。因为开始时，1-2代价较大，而1-3代价小，导致算法在当前状态下误以为1-3更优，而不知道的是，1-3虽然当前优，但后边的代价很大；而1-2虽当前劣，但后面代价很小。因此我想，如果我们能让算法知道各支路后面大概的代价，那就可以直接搜1-2的支路，节省大量时间

估价函数

- 我们可以设置一个函数，使它能计算出当前状态到目标状态的估计代价。这样，我们仍维护一个堆，维护当前代价+未来还需要的代价的最小值。这样我们就可以粗略计算后面对当前的影响，从而更有可能进入正确的分支，提高搜索效率。
- 估价函数的设计有一个重要准则：
- **预估代价≦实际代价**
- 这是显然的，因为如果预估了较大的代价，那么真正的最优解会因为当前代价+预估代价太大而无法从堆中取出，从而让错误的状态一直扩展，得到错解。而如果往小估价，即使错解先扩展，由于它不是最优的，所以在某一时刻，它的当前代价一定大于最优解。而由于最优路径上的状态的当前代价+估计代价≤最优解，所以此时该状态一定会被扩展，我们就能快速得到最优解。这就是A*算法，我们来看一道例题

【模板】k 短路 / [SDOI2010] 魔法猪学院

- 给出一张边权为正实数的有向图，求最多能选出多少条从1到n的不同路径，使路径长度和不超过给定的正实数E。
- $n \leqq 5000, m \leqq 200000, E \leqq 10000000, e_i \leqq E$.
- 注：对于 k 短路问题，A^* 算法的最坏复杂度是$O(nklogn)$，但A^*算法在大部分情况下是$O(tnlogn)$的，其中t是较小的常数，在本题中可以获得100pts,但过不了，因为有构造的hack数据。事实上，存在使用可持久化可并堆的算法可以做到$O((n+m)logn+klogk)$，感兴趣的同学可以自行学习，这里不做介绍~~我也不会~~，本题主要用来 让大家了解A*的具体实现。

- 显然，我们要选择1~n最短的k条路，所以我们的任务是求出1~n前k短的路径。
- 回顾dijkstra求最短路的过程，当我们从堆中第一次取出这个节点时，就得到了这个节点的最短路。很容易得到推论：从堆中第k次取出这个节点时，就得到了这个节点的k短路。可以用数学归纳法证明。
- 我们只关心1~n的k短路，为了加快搜索效率，我们希望尽量让节点n多被取出，快速逼近答案。于是就可以考虑A*。
- 根据A*的步骤，我们要设计估价函数，及节点i到n的k短路的预估距离。秉承**预估代价≦实际代价**的原则，我们发现，i到n的最短路就是很好的估价函数。所以我们以n为起点，在反图上跑遍dijkstra,记f[i]表示i到n的最短路，那我们的堆就要维护当前已有代价+预估代价最小的节点扩展。详细的，我们可以看代码。
- [记录详情 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)]

IDA*即为估价函数在dfs上的运用，一般和迭代加深搜索配合。IDA*比A*运用更广，且实现更容易，效率更高，也需要满足预估代价≤实际代价的原则。我们直接来看题：

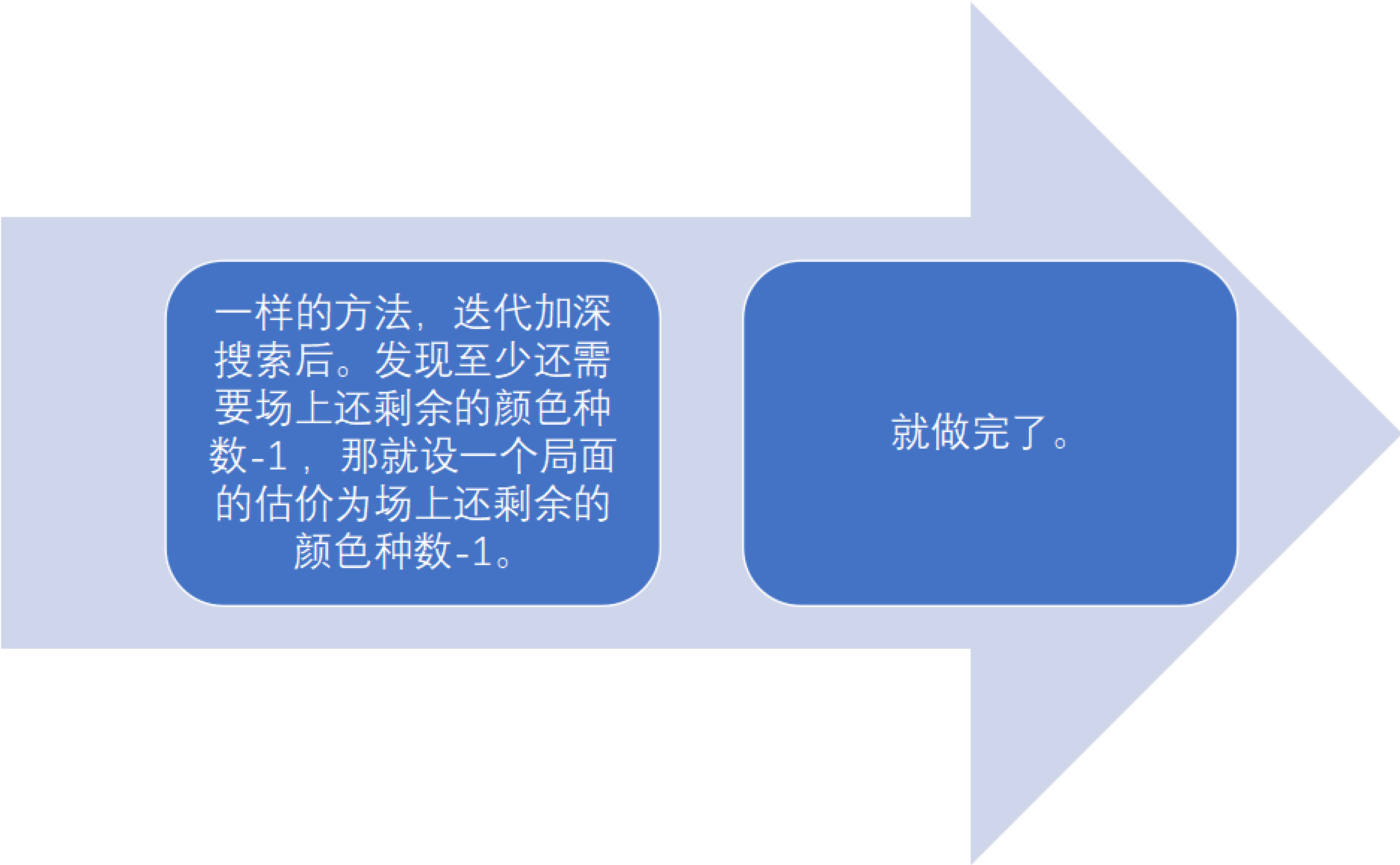
[P2324 [SCOI2005] 骑士精神 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)]()

- 注意观察输出格式那句话。

- 我们枚举空格往哪跳，而不是马往哪跳。这显然等价，且方便搜索。
- 使用迭代加深搜索，尝试在限制步数内搜寻答案，若超过限制还没搜出来就宣判当前无解。
- 这样会超时，我们考虑用IDA*。我们要设计的估价函数必须小于等于实际步数。可以发现，每一次移动，最多让一只马回到正确的位置，也就是说，对于一个局面，如果有k只马不在正确的位置上，那么至少需要再操作k次才能到目标状态。于是，我们可以再把估价函数定义为一个局面上站错位置的马的个数。若已操作步数+估价步数超过限制，直接返回当前无解。
- 记录详情 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)

UVA1505 Flood-it!

- [Flood-It (unixpapa.com)]


一样的方法，迭代加深搜索后。发现至少还需要场上还剩余的颜色种数-1，那就设一个局面的估价为场上还剩余的颜色种数-1。
就做完了。

感谢聆听！！！QAQ