NP-Hard问题求解方法杂谈

福建省福州第一中学 钟知闲

February 5, 2018

目录

NP-Hard问题概述

- P问题：多项式问题，即在图灵机上（下同），可在多项式时间内求解的问题
- NP问题：非确定性多项式问题，即可以在多项式时间内验证解的问题
- NPC问题：一类NP问题，满足所有NP问题都可以在多项式时间内规约到该问题
- NP-Hard问题：一类问题，满足所有NP问题都可以在多项式时间内规约到该问题；通常认为NP-Hard问题无法在多项式时间内求解
- NPC问题是NP问题和NP-Hard问题的交集

这里的验证解指问题的判定形式

NP-Hard问题概述

- P问题：多项式问题，即在图灵机上（下同），可在多项式时间内求解的问题
- NP问题：非确定性多项式问题，即可以在多项式时间内验证解的问题
- NPC问题：一类NP问题，满足所有NP问题都可以在多项式时间内规约到该问题
- NP-Hard问题：一类问题，满足所有NP问题都可以在多项式时间内规约到该问题；通常认为NP-Hard问题无法在多项式时间内求解
- NPC问题是NP问题和NP-Hard问题的交集

这里的验证解指问题的判定形式

NP-Hard问题概述

- P问题：多项式问题，即在图灵机上（下同），可在多项式时间内求解的问题
- NP问题：非确定性多项式问题，即可以在多项式时间内验证解的问题
- NPC问题：一类NP问题，满足所有NP问题都可以在多项式时间内规约到该问题
- NP-Hard问题：一类问题，满足所有NP问题都可以在多项式时间内规约到该问题；通常认为NP-Hard问题无法在多项式时间内求解
- NPC问题是NP问题和NP-Hard问题的交集

这里的验证解指问题的判定形式

NP-Hard问题概述

- P问题：多项式问题，即在图灵机上（下同），可在多项式时间内求解的问题
- NP问题：非确定性多项式问题，即可以在多项式时间内验证解的问题
- NPC问题：一类NP问题，满足所有NP问题都可以在多项式时间内规约到该问题
- NP-Hard问题：一类问题，满足所有NP问题都可以在多项式时间内规约到该问题；通常认为NP-Hard问题无法在多项式时间内求解
- NPC问题是NP问题和NP-Hard问题的交集

这里的验证解指问题的判定形式

NP-Hard问题概述

- P问题：多项式问题，即在图灵机上（下同），可在多项式时间内求解的问题
- NP问题：非确定性多项式问题，即可以在多项式时间内验证解的问题
- NPC问题：一类NP问题，满足所有NP问题都可以在多项式时间内规约到该问题
- NP-Hard问题：一类问题，满足所有NP问题都可以在多项式时间内规约到该问题；通常认为NP-Hard问题无法在多项式时间内求解
- NPC问题是NP问题和NP-Hard问题的交集

这里的验证解指问题的判定形式

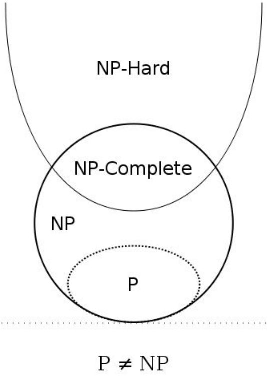
NP-Hard问题概述

- P问题：多项式问题，即在图灵机上（下同），可在多项式时间内求解的问题
- NP问题：非确定性多项式问题，即可以在多项式时间内验证解的问题
- NPC问题：一类NP问题，满足所有NP问题都可以在多项式时间内规约到该问题
- NP-Hard问题：一类问题，满足所有NP问题都可以在多项式时间内规约到该问题；通常认为NP-Hard问题无法在多项式时间内求解
- NPC问题是NP问题和NP-Hard问题的交集

这里的验证解指问题的判定形式

NP-Hard问题概述

通常认为$P \neq NP$



常见NP-Hard问题

- 背包问题：给定一组物品的重量和价值，选出一些物品使得重量之和不超过C且价值之和最大
- 独立集问题：给定n阶无向图$G=(V,E)$，求V的最大的子集S，使S中任意两点不相邻
- 旅行商问题：给定带权完全图G，求一条经过每个点恰好一次的路径，使得路径的权值和最小
- ……

常见NP-Hard问题

- 背包问题：给定一组物品的重量和价值，选出一些物品使得重量之和不超过C且价值之和最大
- 独立集问题：给定n阶无向图$G = (V, E)$，求V的最大的子集S，使S中任意两点不相邻
- 旅行商问题：给定带权完全图G，求一条经过每个点恰好一次的路径，使得路径的权值和最小
- ……

常见NP-Hard问题

- 背包问题：给定一组物品的重量和价值，选出一些物品使得重量之和不超过C且价值之和最大
- 独立集问题：给定n阶无向图$G=(V,E)$，求V的最大的子集S，使S中任意两点不相邻
- 旅行商问题：给定带权完全图G，求一条经过每个点恰好一次的路径，使得路径的权值和最小
- ……

常见NP-Hard问题

- 背包问题：给定一组物品的重量和价值，选出一些物品使得重量之和不超过C且价值之和最大
- 独立集问题：给定n阶无向图$G=(V,E)$，求V的最大的子集S，使S中任意两点不相邻
- 旅行商问题：给定带权完全图G，求一条经过每个点恰好一次的路径，使得路径的权值和最小
- ……

求解NP-Hard问题的常用策略

由于NP-Hard问题难以高效解决，求解NP-Hard问题的策略大致可以分为两个方向：

- 精确算法：保证解的最优性，在此基础上优化算法运行效率（仍然是指数级的）
- 近似算法：用时间可以接受的算法，求出尽量优的解

求解NP-Hard问题的常用策略

由于NP-Hard问题难以高效解决，求解NP-Hard问题的策略大致可以分为两个方向：

- 精确算法：保证解的最优性，在此基础上优化算法运行效率（仍然是指数级的）
- 近似算法：用时间可以接受的算法，求出尽量优的解

求解NP-Hard问题的常用策略

由于NP-Hard问题难以高效解决，求解NP-Hard问题的策略大致可以分为两个方向：

- 精确算法：保证解的最优性，在此基础上优化算法运行效率（仍然是指数级的）
- 近似算法：用时间可以接受的算法，求出尽量优的解

精确算法

大多数OI题目必须求出最优解才能满分，应使用精确算法

精确算法的瓶颈在于时间复杂度

精确算法

大多数OI题目必须求出最优解才能满分，应使用精确算法
精确算法的瓶颈在于时间复杂度

暴力搜索

暴力枚举问题的所有解，效率较低

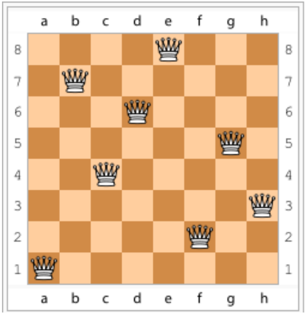
可以根据问题设计剪枝来优化搜索效率

暴力搜索

暴力枚举问题的所有解，效率较低
可以根据问题设计剪枝来优化搜索效率

例：n皇后问题

在$n \times n$的棋盘上放n个皇后，使得任意两个皇后都不在同一行、行或斜线上。



例：n皇后问题

3101: N皇后

Time Limit: 10 Sec Memory Limit: 128 MBSec Special Judge

Submit: 70 Solved: 32

[Submit][Status]

Description

n*n的棋盘,在上面摆下n个皇后，使其两两间不能相互攻击...

Input

一个数n

Output

第i行表示在第i行第几列放置皇后

我会$O(n)$！

好，问题改一改……求所有的方案！

例：n皇后问题


3101: N皇后

Time Limit: 10 Sec Memory Limit: 128 MBSec Special Judge

Submit: 70 Solved: 32

[Submit][Status]

Description

n*n的棋盘,在上面摆下n个皇后，使其两两间不能相互攻击...

Input

一个数n

Output

第i行表示在第i行第几列放置皇后

我会$O(n)$！

好，问题改一改......求所有的方案！

例：n皇后问题

不同的暴搜还是有点区别的

怎样的剪枝优化比较优秀呢?

- 剪枝原则：正确性，准确性，高效性

例：n皇后问题

不同的暴搜还是有点区别的
怎样的剪枝优化比较优秀呢?

- 剪枝原则：正确性，准确性，高效性

例：n皇后问题

- 压位大法

```
#include<cstdio>
int n,u,ans;
void dfs(unsigned s1,unsigned s2,unsigned s3){
    if(s1==u)++ans;
    else for(unsigned t=u&~(s1|s2|s3),i;t;t-=i)
        i=t&-t,dfs(s1|i,(s2|i)<<1,(s3|i)>>1);
}
int main(){
    scanf("%d",&n);
    u=(1<<n)-1;
    dfs(0,0,0);
    printf("%d\n",ans);
}
```

例：数独问题

在空格上填入1-9的数字。使1-9每个数字在每一行、每一列和每一宫中都只出现一次。

8								
		3	6					
	7			9		2		
	5				7			
				4	5	7		
			1				3	
		1					6	8
		8	5				1	
	9					4		

Dancing Links

数独问题可以转为精确覆盖问题
构造一个324列729行的矩阵，每列对应一个约束：

- 第$1\sim81$列：表示每个格子是否有填数
- 第$82\sim162$列：表示每行是否有出现$1\sim9$
- 第$163\sim243$列：表示每列是否有出现$1\sim9$
- 第$244\sim324$列：表示每宫是否有出现$1\sim9$

每个格子对应矩阵中的9行，每行包含4个1，其余位置为0，表示在该空位填$1\sim9$占据的格子、行、列、宫
对于已填的格子，强制选取对应的行

Dancing Links

数独问题可以转为精确覆盖问题
构造一个324列729行的矩阵，每列对应一个约束：

- 第$1\sim81$列：表示每个格子是否有填数
- 第$82\sim162$列：表示每行是否有出现$1\sim9$
- 第$163\sim243$列：表示每列是否有出现$1\sim9$
- 第$244\sim324$列：表示每宫是否有出现$1\sim9$

每个格子对应矩阵中的9行，每行包含4个1，其余位置为0，表示在该空位填$1\sim9$占据的格子、行、列、宫
对于已填的格子，强制选取对应的行

Dancing Links

数独问题可以转为精确覆盖问题
构造一个324列729行的矩阵，每列对应一个约束：

- 第$1\sim81$列：表示每个格子是否有填数
- 第$82\sim162$列：表示每行是否有出现$1\sim9$
- 第$163\sim243$列：表示每列是否有出现$1\sim9$
- 第$244\sim324$列：表示每宫是否有出现$1\sim9$

每个格子对应矩阵中的9行，每行包含4个1，其余位置为0，表示在该空位填$1\sim9$占据的格子、行、列、宫
对于已填的格子，强制选取对应的行

Dancing Links

数独问题可以转为精确覆盖问题
构造一个324列729行的矩阵，每列对应一个约束：

- 第$1 \sim 81$列：表示每个格子是否有填数
- 第$82 \sim 162$列：表示每行是否有出现$1 \sim 9$
- 第$163 \sim 243$列：表示每列是否有出现$1 \sim 9$
- 第$244 \sim 324$列：表示每宫是否有出现$1 \sim 9$

每个格子对应矩阵中的9行，每行包含4个1，其余位置为0，表示在该空位填$1 \sim 9$占据的格子、行、列、宫
对于已填的格子，强制选取对应的行

Dancing Links

数独问题可以转为精确覆盖问题
构造一个324列729行的矩阵，每列对应一个约束：

- 第$1 \sim 81$列：表示每个格子是否有填数
- 第$82 \sim 162$列：表示每行是否有出现$1 \sim 9$
- 第$163 \sim 243$列：表示每列是否有出现$1 \sim 9$
- 第$244 \sim 324$列：表示每宫是否有出现$1 \sim 9$

每个格子对应矩阵中的9行，每行包含4个1，其余位置为0，表示在该空位填$1 \sim 9$占据的格子、行、列、宫
对于已填的格子，强制选取对应的行

Dancing Links

数独问题可以转为精确覆盖问题
构造一个324列729行的矩阵，每列对应一个约束：

- 第$1\sim81$列：表示每个格子是否有填数
- 第$82\sim162$列：表示每行是否有出现$1\sim9$
- 第$163\sim243$列：表示每列是否有出现$1\sim9$
- 第$244\sim324$列：表示每宫是否有出现$1\sim9$

每个格子对应矩阵中的9行，每行包含4个1，其余位置为0，表示在该空位填$1\sim9$占据的格子、行、列、宫
对于已填的格子，强制选取对应的行

Dancing Links

问题转化为给定01矩阵A，选取A的若干行，使得每列恰有一个1

DLX算法：

- 选择A中1最少的一列j
- 枚举满足$A_{i,j}=1$的行i，对所有$A_{i,k}=1$的k，删除满足$A_{l,k}=1$的行l
- 当矩阵为空时说明找到解
- 如果找到解则输出，否则回溯到原来矩阵，选择下一个第j列为1的行，继续递归；
- 如果删除每一行都无解，返回无解

由于矩阵通常很稀疏，用双向链表实现，优化效率

Dancing Links

问题转化为给定01矩阵A，选取A的若干行，使得每列恰有一个1

DLX算法：

- 选择A中1最少的一列j
- 枚举满足$A_{i,j}=1$的行i，对所有$A_{i,k}=1$的k，删除满足$A_{l,k}=1$的行l
- 当矩阵为空时说明找到解
- 如果找到解则输出，否则回溯到原来矩阵，选择下一个第j列为1的行，继续递归；
- 如果删除每一行都无解，返回无解

由于矩阵通常很稀疏，用双向链表实现，优化效率

Dancing Links

问题转化为给定01矩阵A，选取A的若干行，使得每列恰有一个1

DLX算法：

- 选择A中1最少的一列j
- 枚举满足$A_{i,j}=1$的行i，对所有$A_{i,k}=1$的k，删除满足$A_{l,k}=1$的行l
- 当矩阵为空时说明找到解
- 如果找到解则输出，否则回溯到原来矩阵，选择下一个第j列为1的行，继续递归；
- 如果删除每一行都无解，返回无解

由于矩阵通常很稀疏，用双向链表实现，优化效率

Dancing Links

问题转化为给定01矩阵A，选取A的若干行，使得每列恰有一个1

DLX算法：

- 选择A中1最少的一列j
- 枚举满足$A_{i,j}=1$的行i，对所有$A_{i,k}=1$的k，删除满足$A_{l,k}=1$的行l
- 当矩阵为空时说明找到解
- 如果找到解则输出，否则回溯到原来矩阵，选择下一个第j列为1的行，继续递归；
- 如果删除每一行都无解，返回无解

由于矩阵通常很稀疏，用双向链表实现，优化效率

Dancing Links

问题转化为给定01矩阵A，选取A的若干行，使得每列恰有一个1

DLX算法：

- 选择A中1最少的一列j
- 枚举满足$A_{i,j}=1$的行i，对所有$A_{i,k}=1$的k，删除满足$A_{l,k}=1$的行l
- 当矩阵为空时说明找到解
- 如果找到解则输出，否则回溯到原来矩阵，选择下一个第j列为1的行，继续递归；
- 如果删除每一行都无解，返回无解

由于矩阵通常很稀疏，用双向链表实现，优化效率

Dancing Links

问题转化为给定01矩阵A，选取A的若干行，使得每列恰有一个1

DLX算法：

- 选择A中1最少的一列j
- 枚举满足$A_{i,j}=1$的行i，对所有$A_{i,k}=1$的k，删除满足$A_{l,k}=1$的行l
- 当矩阵为空时说明找到解
- 如果找到解则输出，否则回溯到原来矩阵，选择下一个第j列为1的行，继续递归；
- 如果删除每一行都无解，返回无解

由于矩阵通常很稀疏，用双向链表实现，优化效率

Dancing Links

问题转化为给定01矩阵A，选取A的若干行，使得每列恰有一个1

DLX算法：

- 选择A中1最少的一列j
- 枚举满足$A_{i,j} = 1$的行i，对所有$A_{i,k} = 1$的k，删除满足$A_{l,k} = 1$的行l
- 当矩阵为空时说明找到解
- 如果找到解则输出，否则回溯到原来矩阵，选择下一个第j列为1的行，继续递归；
- 如果删除每一行都无解，返回无解

由于矩阵通常很稀疏，用双向链表实现，优化效率

分支定界法

分支定界可以认为是一种启发式的搜索剪枝

设解空间为S，考虑求解$x \in S$最小化函数$f(x)$的值

- 维护一个数据结构维护若干个解的一部分，每个对应S的一个子集S_i，$\bigcup S_i = S$，同时维护当前最优解x_0
- 每次从数据结构中取出一个元素N，如果N包含确定的解x，即$N = \{x\}$，则当$f(x) < f(x_0)$时，用x更新x_0
- 否则将N拆分为若干部分X_i；定义函数$h(X_i)$，满足$h(X_i) \leq f(x), \forall x \in X_i$；如果$h(X_i) \geq f(x_0)$则忽略$X_i$，否则将$X_i$加入数据结构

数据结构可以选择栈、队列、优先队列等

分支定界法

分支定界可以认为是一种启发式的搜索剪枝
设解空间为S，考虑求解$x \in S$最小化函数$f(x)$的值

- 维护一个数据结构维护若干个解的一部分，每个对应S的一个子集S_i，$\bigcup S_i = S$，同时维护当前最优解x_0
- 每次从数据结构中取出一个元素N，如果N包含确定的解x，即$N = \{x\}$，则当$f(x) < f(x_0)$时，用x更新x_0
- 否则将N拆分为若干部分X_i；定义函数$h(X_i)$，满足$h(X_i) \le f(x), \forall x \in X_i$；如果$h(X_i) \ge f(x_0)$则忽略$X_i$，否则将$X_i$加入数据结构

数据结构可以选择栈、队列、优先队列等

分支定界法

分支定界可以认为是一种启发式的搜索剪枝
设解空间为S，考虑求解$x \in S$最小化函数$f(x)$的值

- 维护一个数据结构维护若干个解的一部分，每个对应S的一个子集S_i，$\bigcup S_i = S$，同时维护当前最优解x_0
- 每次从数据结构中取出一个元素N，如果N包含确定的解x，即$N = \{x\}$，则当$f(x) < f(x_0)$时，用x更新x_0
- 否则将N拆分为若干部分X_i；定义函数$h(X_i)$，满足$h(X_i) \leq f(x), \forall x \in X_i$；如果$h(X_i) \geq f(x_0)$则忽略$X_i$，否则将$X_i$加入数据结构

数据结构可以选择栈、队列、优先队列等

分支定界法

分支定界可以认为是一种启发式的搜索剪枝
设解空间为S，考虑求解$x \in S$最小化函数$f(x)$的值

- 维护一个数据结构维护若干个解的一部分，每个对应S的一个子集S_i，$\bigcup S_i = S$，同时维护当前最优解x_0
- 每次从数据结构中取出一个元素N，如果N包含确定的解x，即$N = \{x\}$，则当$f(x) < f(x_0)$时，用x更新x_0
- 否则将N拆分为若干部分X_i；定义函数$h(X_i)$，满足$h(X_i) \le f(x), \forall x \in X_i$；如果$h(X_i) \ge f(x_0)$则忽略$X_i$，否则将$X_i$加入数据结构

数据结构可以选择栈、队列、优先队列等

分支定界法

分支定界可以认为是一种启发式的搜索剪枝
设解空间为S，考虑求解$x \in S$最小化函数$f(x)$的值

- 维护一个数据结构维护若干个解的一部分，每个对应S的一个子集S_i，$\bigcup S_i = S$，同时维护当前最优解x_0
- 每次从数据结构中取出一个元素N，如果N包含确定的解x，即$N = \{x\}$，则当$f(x) < f(x_0)$时，用x更新x_0
- 否则将N拆分为若干部分X_i；定义函数$h(X_i)$，满足$h(X_i) \leq f(x), \forall x \in X_i$；如果$h(X_i) \geq f(x_0)$则忽略$X_i$，否则将$X_i$加入数据结构

数据结构可以选择栈、队列、优先队列等

分支定界法

分支定界可以认为是一种启发式的搜索剪枝
设解空间为S，考虑求解$x \in S$最小化函数$f(x)$的值

- 维护一个数据结构维护若干个解的一部分，每个对应S的一个子集S_i，$\bigcup S_i = S$，同时维护当前最优解x_0
- 每次从数据结构中取出一个元素N，如果N包含确定的解x，即$N = \{x\}$，则当$f(x) < f(x_0)$时，用x更新x_0
- 否则将N拆分为若干部分X_i；定义函数$h(X_i)$，满足$h(X_i) \le f(x), \forall x \in X_i$；如果$h(X_i) \ge f(x_0)$则忽略$X_i$，否则将$X_i$加入数据结构

数据结构可以选择栈、队列、优先队列等

分支定界法

可以形象化地理解：求状态图中s到t的最短路$d(s)$，可以对结点设计估价函数$h(x)$，满足$h(x) \le d(x)$，设当前搜索状态s到x的距离为$g(x)$，如果目前搜到的最优解

$$ans \le g(x) + h(x)$$

则该状态继续扩展不影响答案，可以剪枝

估价函数$h(x)$越接近$d(x)$则剪枝效果越好，也是解决问题的关键

分支定界法

可以形象化地理解：求状态图中s到t的最短路$d(s)$，可以对结点设计估价函数$h(x)$，满足$h(x) \leq d(x)$，设当前搜索状态s到x的距离为$g(x)$，如果目前搜到的最优解

$$ans \leq g(x) + h(x)$$

则该状态继续扩展不影响答案，可以剪枝

估价函数$h(x)$越接近$d(x)$则剪枝效果越好，也是解决问题的关键

分支定界法

A*：用优先队列存储状态，同时记录所有经过的状态，每次选取$g(x)+h(x)$最小的状态扩展

- 可以认为是优化的BFS
- 每个状态只会访问一次，但空间消耗大

IDA*：用栈存储状态，不记录所有经过的状态，而是先枚举搜索深度d，在$g(x)+h(x)>d$时剪枝

- 可以认为是优化的DFS
- 空间消耗小，但状态重复访问多

分支定界法

A*：用优先队列存储状态，同时记录所有经过的状态，每次选取$g(x)+h(x)$最小的状态扩展

- 可以认为是优化的BFS
- 每个状态只会访问一次，但空间消耗大

IDA*：用栈存储状态，不记录所有经过的状态，而是先枚举搜索深度d，在$g(x)+h(x)>d$时剪枝

- 可以认为是优化的DFS
- 空间消耗小，但状态重复访问多

分支定界法

A*：用优先队列存储状态，同时记录所有经过的状态，每次选取$g(x)+h(x)$最小的状态扩展

- 可以认为是优化的BFS
- 每个状态只会访问一次，但空间消耗大

IDA*：用栈存储状态，不记录所有经过的状态，而是先枚举搜索深度d，在$g(x)+h(x)>d$时剪枝

- 可以认为是优化的DFS
- 空间消耗小，但状态重复访问多

分支定界法

A*：用优先队列存储状态，同时记录所有经过的状态，每次选取$g(x)+h(x)$最小的状态扩展

- 可以认为是优化的BFS
- 每个状态只会访问一次，但空间消耗大

IDA*：用栈存储状态，不记录所有经过的状态，而是先枚举搜索深度d，在$g(x)+h(x)>d$时剪枝

- 可以认为是优化的DFS
- 空间消耗小，但状态重复访问多

分支定界法

A*：用优先队列存储状态，同时记录所有经过的状态，每次选取$g(x)+h(x)$最小的状态扩展

- 可以认为是优化的BFS
- 每个状态只会访问一次，但空间消耗大

IDA*：用栈存储状态，不记录所有经过的状态，而是先枚举搜索深度d，在$g(x)+h(x)>d$时剪枝

- 可以认为是优化的DFS
- 空间消耗小，但状态重复访问多

分支定界法

A*：用优先队列存储状态，同时记录所有经过的状态，每次选取$g(x)+h(x)$最小的状态扩展

- 可以认为是优化的BFS
- 每个状态只会访问一次，但空间消耗大

IDA*：用栈存储状态，不记录所有经过的状态，而是先枚举搜索深度d，在$g(x)+h(x)>d$时剪枝

- 可以认为是优化的DFS
- 空间消耗小，但状态重复访问多

动态规划法

不少问题中，搜索会经常搜到重复的状态

使用记忆化的思想，搜索时把这些状态记录下来重复利用，以提高效率

有时用记忆化搜索实现，即搜索时开个数组或hash表记录每个状态的结果，如果搜到已有的状态直接返回

动态规划法

不少问题中，搜索会经常搜到重复的状态

使用记忆化的思想，搜索时把这些状态记录下来重复利用，以提高效率

有时用记忆化搜索实现，即搜索时开个数组或hash表记录每个状态的结果，如果搜到已有的状态直接返回

动态规划法

不少问题中，搜索会经常搜到重复的状态

使用记忆化的思想，搜索时把这些状态记录下来重复利用，以提高效率

有时用记忆化搜索实现，即搜索时开个数组或hash表记录每个状态的结果，如果搜到已有的状态直接返回

例：背包问题

- 0-1背包问题：给定大小为n的物品集合中每个物品的重量w_i和价值v_i，以及一个容量C，选出一个物品的子集S，满足总重量$\sum_{i \in S} w_i \leq C$，最大化总价值$V = \sum_{i \in S} v_i$
- 完全背包问题：每个物品有无穷多份，其余与0-1背包一致

当C是整数时，DP可以做到$O(nC)$
（然而这是指数级的哦！）

例：背包问题

- 0-1背包问题：给定大小为n的物品集合中每个物品的重量w_i和价值v_i，以及一个容量C，选出一个物品的子集S，满足总重量$\sum_{i \in S} w_i \le C$，最大化总价值$V = \sum_{i \in S} v_i$
- 完全背包问题：每个物品有无穷多份，其余与0-1背包一致

当C是整数时，DP可以做到$O(nC)$
（然而这是指数级的哦！）

例：背包问题

- 0-1背包问题：给定大小为n的物品集合中每个物品的重量w_i和价值v_i，以及一个容量C，选出一个物品的子集S，满足总重量$\sum_{i \in S} w_i \le C$，最大化总价值$V = \sum_{i \in S} v_i$
- 完全背包问题：每个物品有无穷多份，其余与0-1背包一致

当C是整数时，DP可以做到$O(nC)$

（然而这是指数级的哦！）

例：背包问题

- 0-1背包问题：给定大小为n的物品集合中每个物品的重量w_i和价值v_i，以及一个容量C，选出一个物品的子集S，满足总重量$\sum_{i \in S} w_i \le C$，最大化总价值$V = \sum_{i \in S} v_i$
- 完全背包问题：每个物品有无穷多份，其余与0-1背包一致

当C是整数时，DP可以做到$O(nC)$
（然而这是指数级的哦！）

例：图染色问题

给定n阶无向图G，求G的色数

- $O(n!)$
- $O(3^n)$?
- $O(2^n n)$?

例：图染色问题

给定n阶无向图G，求G的色数

- $O(n!)$
- $O(3^n)$?
- $O(2^n n)$?

例：图染色问题

给定n阶无向图G，求G的色数

- $O(n!)$
- $O(3^n)$?
- $O(2^n n)$?

例：图染色问题

给定n阶无向图G，求G的色数

- $O(n!)$
- $O(3^n)$?
- $O(2^n n)$?

例：图染色问题

考虑判断G的色数是否不大于k，即能否用k个独立集覆盖$V(G)$

- 首先处理出G的所有独立集$\{I_1, I_2, \cdots, I_m\}$
- 不妨设$V(G) = \{1, 2, \cdots, n\}$，记$f(i, S)$表示有多少个独立集$T \subseteq S$满足$S - T \subseteq \{1, 2, \cdots, i\}$，则

$$f(i, S) = f(i-1, S) + [i \in S] f(i-1, S - \{i\})$$

- 容斥可得用k个独立集覆盖$V(G)$的方案数为

$$\sum_{S \subseteq V(G)} (-1)^{|S|} f(n, V(G) - S)^k$$

例：图染色问题

考虑判断G的色数是否不大于k，即能否用k个独立集覆盖$V(G)$

- 首先处理出G的所有独立集$\{I_1, I_2, \cdots, I_m\}$
- 不妨设$V(G) = \{1, 2, \cdots, n\}$，记$f(i, S)$表示有多少个独立集$T \subseteq S$满足$S - T \subseteq \{1, 2, \cdots, i\}$，则

$$f(i, S) = f(i-1, S) + [i \in S] f(i-1, S - \{i\})$$

- 容斥可得用k个独立集覆盖$V(G)$的方案数为

$$\sum_{S \subseteq V(G)} (-1)^{|S|} f(n, V(G) - S)^k$$

例：图染色问题

考虑判断G的色数是否不大于k，即能否用k个独立集覆盖$V(G)$

- 首先处理出G的所有独立集$\{I_1, I_2, \cdots, I_m\}$
- 不妨设$V(G) = \{1, 2, \cdots, n\}$，记$f(i, S)$表示有多少个独立集$T \subseteq S$满足$S - T \subseteq \{1, 2, \cdots, i\}$，则

$$f(i, S) = f(i-1, S) + [i \in S] f(i-1, S - \{i\})$$

- 容斥可得用k个独立集覆盖$V(G)$的方案数为

$$\sum_{S \subseteq V(G)} (-1)^{|S|} f(n, V(G) - S)^k$$

例：图染色问题

考虑判断G的色数是否不大于k，即能否用k个独立集覆盖$V(G)$

- 首先处理出G的所有独立集$\{I_1, I_2, \cdots, I_m\}$
- 不妨设$V(G) = \{1, 2, \cdots, n\}$，记$f(i, S)$表示有多少个独立集$T \subseteq S$满足$S - T \subseteq \{1, 2, \cdots, i\}$，则

$$f(i, S) = f(i-1, S) + [i \in S] f(i-1, S - \{i\})$$

- 容斥可得用k个独立集覆盖$V(G)$的方案数为

$$\sum_{S \subseteq V(G)} (-1)^{|S|} f(n, V(G) - S)^k$$

例：图染色问题

预处理f，枚举$k = 1, 2, \cdots, n$直到方案数非0为止

可以在模随机质数意义下搞

复杂度$O(2^n n)$

例：图染色问题

预处理f，枚举$k = 1, 2, \cdots, n$直到方案数非0为止

可以在模随机质数意义下搞

复杂度$O(2^n n)$

例：图染色问题

预处理f，枚举$k = 1, 2, \cdots, n$直到方案数非0为止
可以在模随机质数意义下搞
复杂度$O(2^n n)$

例：带权独立集问题

给定无向图$G=(V,E)$和权函数$w:V\rightarrow\mathbf{N}^*$，求一个$S\subseteq V$，满足$S$中的点两两不相邻，最大化$\sum_{v\in S}w(v)$

朴素搜索——$O(2^n)$

例：带权独立集问题

给定无向图$G=(V,E)$和权函数$w:V\to\mathbf{N}^*$，求一个$S\subseteq V$，满足S中的点两两不相邻，最大化$\sum_{v\in S}w(v)$

朴素搜索——$O(2^n)$

例：带权独立集问题

只枚举极大独立集，即独立集I，满足$\forall v \in V$，$I + \{v\}$不是独立集

极大独立集的个数比独立集的个数少得多

怎样的图能把极大独立集的个数卡到尽量多？

例：带权独立集问题

只枚举极大独立集，即独立集I，满足$\forall v \in V$，$I+\{v\}$不是独立集

极大独立集的个数比独立集的个数少得多

怎样的图能把极大独立集的个数卡到尽量多?

例：带权独立集问题

只枚举极大独立集，即独立集I，满足$\forall v \in V$，$I + \{v\}$不是独立集

极大独立集的个数比独立集的个数少得多

怎样的图能把极大独立集的个数卡到尽量多？

例：带权独立集问题

极大独立集的个数上界为$O(3^{\frac{n}{3}})$

Bron-Kerbosch算法——$O(3^{\frac{n}{3}})$枚举所有极大独立集

例：带权独立集问题

极大独立集的个数上界为$O(3^{\frac{n}{3}})$
Bron-Kerbosch算法——$O(3^{\frac{n}{3}})$枚举所有极大独立集

例：带权独立集问题

极大独立集的个数上界为$O(3^{\frac{n}{3}})$
Bron-Kerbosch算法——$O(3^{\frac{n}{3}})$枚举所有极大独立集

例：带权独立集问题

BronKerbosch(R,P,X)：列举所有包含R中所有结点、P中部分节点且不包含X中任何结点的极大独立集

算法 1 BronKerbosch(R,P,X)

if $P=X=\varnothing$ **then**
  print R
end if
选择结点 $u\in P\cup X$，使得 $|P\cap(\{u\}\cup N(u))|$ 最小
for all $v\in P\cap(\{u\}\cup N(u))$ **do**
  BronKerbosch$(R\cup\{v\},P-(\{v\}\cup N(v)),X-(\{v\}\cup N(v)))$
  $P\leftarrow P-\{v\}$
  $X\leftarrow X\cup\{v\}$
end for=0

关键：Pivot——

- 选择结点$u\in P\cup X$使得$|P\cup(\{u\}\cup N(u))|$最小
- 枚举$\{u\}\cup N(u)$中属于独立集的第一个结点

例：带权独立集问题

BronKerbosch(R, P, X)：列举所有包含R中所有结点、P中部分节点且不包含X中任何结点的极大独立集

算法 1 BronKerbosch(R, P, X)

if $P = X = \varnothing$ **then**
  print R
end if
选择结点 $u \in P \cup X$，使得 $|P \cap (\{u\} \cup N(u))|$ 最小
for all $v \in P \cap (\{u\} \cup N(u))$ **do**
  BronKerbosch$(R \cup \{v\}, P - (\{v\} \cup N(v)), X - (\{v\} \cup N(v)))$
  $P \leftarrow P - \{v\}$
  $X \leftarrow X \cup \{v\}$
end for=0

关键：Pivot——

- 选择结点$u \in P \cup X$使得$|P \cup (\{u\} \cup N(u))|$最小
- 枚举$\{u\} \cup N(u)$中属于独立集的第一个结点

例：带权独立集问题

BronKerbosch(R, P, X)：列举所有包含R中所有结点、P中部分节点且不包含X中任何结点的极大独立集

```
算法 1 BronKerbosch(R, P, X)
if P = X = ∅ then
    print R
end if
选择结点 u ∈ P ∪ X，使得 |P ∩ ({u} ∪ N(u))| 最小
for all v ∈ P ∩ ({u} ∪ N(u)) do
    BronKerbosch(R ∪ {v}, P − ({v} ∪ N(v)), X − ({v} ∪ N(v)))
    P ← P − {v}
    X ← X ∪ {v}
end for=0
```

关键：Pivot——

- 选择结点$u \in P \cup X$使得$|P \cup (\{u\} \cup N(u))|$最小
- 枚举$\{u\} \cup N(u)$中属于独立集的第一个结点

例：带权独立集问题

BronKerbosch(R,P,X)：列举所有包含R中所有结点、P中部分节点且不包含X中任何结点的极大独立集

算法 1 BronKerbosch(R,P,X)

1: **if** $P = X = \varnothing$ **then**
2:     **print** R
3: **end if**
4: 选择结点 $u \in P \cup X$，使得 $|P \cap (\{u\} \cup N(u))|$ 最小
5: **for all** $v \in P \cap (\{u\} \cup N(u))$ **do**
6:     BronKerbosch$(R \cup \{v\}, P - (\{v\} \cup N(v)), X - (\{v\} \cup N(v)))$
7:     $P \leftarrow P - \{v\}$
8:     $X \leftarrow X \cup \{v\}$
9: **end for**=0

关键：Pivot——

- 选择结点$u \in P \cup X$使得$|P \cup (\{u\} \cup N(u))|$最小
- 枚举$\{u\} \cup N(u)$中属于独立集的第一个结点

例：带权独立集问题

动态规划——$O(2^{\frac{n}{2}})$

- 记$f(S)$为导出子图$G[S]$的最大权独立集的权值和，$N(v)$表示和v相邻的点集
- 对于$v \in S$和$I \subseteq S$，满足$v \in I$，I是$G[S]$独立集等价于$I - \{v\}$是$G[S - \{v\} - N(v)]$的独立集
- $f(S) = \max\{f(S - \{v\}, f(S - \{v\} - N(v)) + w(v)\}$，$v$为任意$S$中结点

这里v取编号最大的结点，记忆化搜索实现

前$\frac{n}{2}$层搜索最多扩展$2^{\frac{n}{2}}$个，后$\frac{n}{2}$层搜索的结点编号不超过$\frac{n}{2}$，所以复杂度是$O(2^{\frac{n}{2}})$

例：带权独立集问题

动态规划——$O(2^{\frac{n}{2}})$

- 记$f(S)$为导出子图$G[S]$的最大权独立集的权值和，$N(v)$表示和v相邻的点集
- 对于$v \in S$和$I \subseteq S$，满足$v \in I$，I是$G[S]$独立集等价于$I - \{v\}$是$G[S - \{v\} - N(v)]$的独立集
- $f(S) = \max\{f(S - \{v\}, f(S - \{v\} - N(v)) + w(v)\}$，$v$为任意$S$中结点

这里v取编号最大的结点，记忆化搜索实现

前$\frac{n}{2}$层搜索最多扩展$2^{\frac{n}{2}}$个，后$\frac{n}{2}$层搜索的结点编号不超过$\frac{n}{2}$，所以复杂度是$O(2^{\frac{n}{2}})$

例：带权独立集问题

动态规划——$O(2^{\frac{n}{2}})$

- 记$f(S)$为导出子图$G[S]$的最大权独立集的权值和，$N(v)$表示和v相邻的点集
- 对于$v \in S$和$I \subseteq S$，满足$v \in I$，I是$G[S]$独立集等价于$I-\{v\}$是$G[S-\{v\}-N(v)]$的独立集
- $f(S) = \max\{f(S-\{v\}, f(S-\{v\}-N(v)) + w(v)\}$，$v$为任意$S$中结点

这里v取编号最大的结点，记忆化搜索实现

前$\frac{n}{2}$层搜索最多扩展$2^{\frac{n}{2}}$个，后$\frac{n}{2}$层搜索的结点编号不超过$\frac{n}{2}$，所以复杂度是$O(2^{\frac{n}{2}})$

例：带权独立集问题

动态规划——$O(2^{\frac{n}{2}})$

- 记$f(S)$为导出子图$G[S]$的最大权独立集的权值和，$N(v)$表示和v相邻的点集
- 对于$v \in S$和$I \subseteq S$，满足$v \in I$，I是$G[S]$独立集等价于$I - \{v\}$是$G[S - \{v\} - N(v)]$的独立集
- $f(S) = \max\{f(S - \{v\}, f(S - \{v\} - N(v)) + w(v)\}$，$v$为任意$S$中结点

这里v取编号最大的结点，记忆化搜索实现

前$\frac{n}{2}$层搜索最多扩展$2^{\frac{n}{2}}$个，后$\frac{n}{2}$层搜索的结点编号不超过$\frac{n}{2}$，所以复杂度是$O(2^{\frac{n}{2}})$

例：带权独立集问题

动态规划——$O(2^{\frac{n}{2}})$

- 记$f(S)$为导出子图$G[S]$的最大权独立集的权值和，$N(v)$表示和v相邻的点集
- 对于$v \in S$和$I \subseteq S$，满足$v \in I$，I是$G[S]$独立集等价于$I - \{v\}$是$G[S - \{v\} - N(v)]$的独立集
- $f(S) = \max\{f(S - \{v\}, f(S - \{v\} - N(v)) + w(v)\}$，$v$为任意$S$中结点

这里v取编号最大的结点，记忆化搜索实现

前$\frac{n}{2}$层搜索最多扩展$2^{\frac{n}{2}}$个，后$\frac{n}{2}$层搜索的结点编号不超过$\frac{n}{2}$，所以复杂度是$O(2^{\frac{n}{2}})$

例：带权独立集问题

优化：

- 选择度数deg(v)最大的点枚举
- 在记忆化搜索过程中，当图不连通时，对各连通块分别递归处理
- 如果是链或者环，还可以直接计算

理论复杂度约$O(n \cdot 1.38^n)$，但可以跑出$n = 80 \sim 100$的随机图
另外，第三点不加其实对效率没多大影响（为什么？）

例：带权独立集问题

优化：

- 选择度数deg(v)最大的点枚举
- 在记忆化搜索过程中，当图不连通时，对各连通块分别递归处理
- 如果是链或者环，还可以直接计算

理论复杂度约$O(n \cdot 1.38^n)$，但可以跑出$n = 80 \sim 100$的随机图
另外，第三点不加其实对效率没多大影响（为什么？）

例：带权独立集问题

优化：

- 选择度数deg(v)最大的点枚举
- 在记忆化搜索过程中，当图不连通时，对各连通块分别递归处理
- 如果是链或者环，还可以直接计算

理论复杂度约$O(n \cdot 1.38^n)$，但可以跑出$n = 80 \sim 100$的随机图
另外，第三点不加其实对效率没多大影响（为什么？）

例：带权独立集问题

优化：

- 选择度数deg(v)最大的点枚举
- 在记忆化搜索过程中，当图不连通时，对各连通块分别递归处理
- 如果是链或者环，还可以直接计算

理论复杂度约$O(n \cdot 1.38^n)$，但可以跑出$n = 80 \sim 100$的随机图
另外，第三点不加其实对效率没多大影响（为什么？）

例：带权独立集问题

优化：

- 选择度数deg(v)最大的点枚举
- 在记忆化搜索过程中，当图不连通时，对各连通块分别递归处理
- 如果是链或者环，还可以直接计算

理论复杂度约$O(n \cdot 1.38^n)$，但可以跑出$n = 80 \sim 100$的随机图
另外，第三点不加其实对效率没多大影响（为什么？）

例：带权独立集问题

更优的独立集算法?

更优的独立集算法

下面是一个$O(1.1888^n)$的最大基数独立集的论文的一小部分

e. **At most one edge ($B_{i,1}, B_{i,2}$)**

W.l.o.g. we suppose that ($B_{2,1}, B_{2,2}$) is not an edge.

i. **Some $B_{i,j}$ has degree £ 4**

stab(G)=max(1+stab(G-[`N](A$_{i¢}$)),stab¢(G-A$_{i¢}$),(N(A$_{i¢}$),2)))

c3ei=max(4,c9,c1b,c2,c3)*alpha[-4]+max(3,a3e,a3f,a3h)*alpha[-1];(= 0.847188).

ii. **Some $B_{i,j}$ has degree ³ 7**

stab(G)=max(1+stab(G-[`N]($B_{i,j}$)),stab(G-$B_{i,j}$))

c3eii=max(3,c9,c1b,c2)*alpha[-8]+c2*alpha[-1];(= 0.826941).

In the remaining cases we write e for the number of edges ($B_{1,1}, B_{2,j}$) assuming w.l.o.g. that this quantity is at least as great for $B_{1,1}$ as for $B_{1,2}$.

iii. **d($B_{1,1}$)=5**

A. **e=0**

stab(G)=max(1+stab(G-[`N]($A_2$)),stab¢(G-$A_2$,({$A_1$,$B_{2,1}$,$B_{2,2}$},2))) where d($B_{1,1}$) < 5 in G-[`N](A_2) and, in G-A_2, d(A_1)=2 and A_1 has no common neighbours with $B_{2,1}$ or $B_{2,2}$.

c3eiiiA=max(5,c9,c1b,c2,c3,c4)*alpha[-4]+a3hiy*alpha[-1];(= 0.797866).

B. **e > 0**

stab(G)=max(1+stab(G-[`N](A_2)),stab¢(G-A_2,({A_1,$B_{2,1}$,$B_{2,2}$},2))).

where d($B_{1,1}$) £ 3 in G-[`N](A_2)

c3eiiiB=max(4,c9,c1b,c2,c3)*alpha[-4]+max(2,a3f,a3h)*alpha[-1];(= 0.847188).

c3eiii=max(2,c3eiiiA,c3eiiiB);(= 0.847188).

iv. **d($B_{1,1}$)=6**

A. **e £ 1**

stab(G)=max(1+stab(G-[`N]($B_{1,1}$)),stab(G-$B_{1,1}$)) where 1 £ d($A_2$) £ 2 in G-[`N]($B_{1,1}$) and d(A_1)=2 in G-$B_{1,1}$.

c3eivA=max(2,c1b,c2)*alpha[-7]+c2*alpha[-1];(= 0.847267).

B. **e=2**

stab(G)=max(1+stab(G-[`N]($A_2$)),stab¢(G-$A_2$,({$A_1$,$B_{2,1}$,$B_{2,2}$},2))) where d($B_{1,1}$)=3 in G-[`N](A_2).

c3eivB=c3*alpha[-4]+max(2,a3f,a3h)*alpha[-1];(= 0.847188).

c3eiv=max(2,c3eivA,c3eivB);(= 0.847267).

c3e=max(4,c3ei,c3eii,c3eii,c3eiv);(= 0.847267).

有兴趣的同学还是自行研究吧……

更优的独立集算法

下面是一个$O(1.1888^n)$的最大基数独立集的论文的一小部分

e. **At most one edge ($B_{i,1}, B_{i,2}$)**
W.l.o.g. we suppose that ($B_{2,1}, B_{2,2}$) is not an edge.

i. **Some $B_{i,j}$ has degree £ 4**
stab(G)=max(1+stab(G-[`N](A_{i¢})),stab¢(G-A_{i¢}),(N(A_{i¢}),2)))
c3ei=max(4,c9,c1b,c2,c3)*alpha[-4]+max(3,a3e,a3f,a3h)*alpha[-1];(= 0.847188).

ii. **Some $B_{i,j}$ has degree ³ 7**
stab(G)=max(1+stab(G-[`N](B_{i,j})),stab(G-B_{i,j}))
c3eii=max(3,c9,c1b,c2)*alpha[-8]+c2*alpha[-1];(= 0.826941).
In the remaining cases we write e for the number of edges ($B_{1,1}, B_{2,j}$) assuming w.l.o.g. that this quantity is at least as great for $B_{1,1}$ as for $B_{1,2}$.

iii. **d($B_{1,1}$)=5**

A. **e=0**
stab(G)=max(1+stab(G-[`N](A_2)),stab¢(G-A_2,({A_1,B_{2,1},B_{2,2}},2))) where d($B_{1,1}$) < 5 in G-[`N](A_2) and, in G-A_2, d(A_1)=2 and A_1 has no common neighbours with $B_{2,1}$ or $B_{2,2}$.
c3eiiiA=max(5,c9,c1b,c2,c3,c4)*alpha[-4]+a3hiy*alpha[-1];(= 0.797866).

B. **e > 0**
stab(G)=max(1+stab(G-[`N](A_2)),stab¢(G-A_2,({A_1,B_{2,1},B_{2,2}},2))).
where d($B_{1,1}$) £ 3 in G-[`N](A_2)
c3eiiiB=max(4,c9,c1b,c2,c3)*alpha[-4]+max(2,a3f,a3h)*alpha[-1];(= 0.847188).
c3eiii=max(2,c3eiiiA,c3eiiiB);(= 0.847188).

iv. **d($B_{1,1}$)=6**

A. **e £ 1**
stab(G)=max(1+stab(G-[`N](B_{1,1})),stab(G-B_{1,1})) where 1 £ d($A_2$) £ 2 in G-[`N](B_{1,1}) and d(A_1)=2 in G-$B_{1,1}$.
c3eivA=max(2,c1b,c2)*alpha[-7]+c2*alpha[-1];(= 0.847267).

B. **e=2**
stab(G)=max(1+stab(G-[`N](A_2)),stab¢(G-A_2,({A_1,B_{2,1},B_{2,2}},2))) where d($B_{1,1}$)=3 in G-[`N](A_2).
c3eivB=c3*alpha[-4]+max(2,a3f,a3h)*alpha[-1];(= 0.847188).
c3eiv=max(2,c3eivA,c3eivB);(= 0.847267).
c3e=max(4,c3ei,c3eii,c3eii,c3eiv);(= 0.847267).

有兴趣的同学还是自行研究吧......

TIPS

WC2013第3题（小Q运动季）的测试点10是最大基数独立集问题，可以试试这个点你能多久跑出最优解

例：Mountains

（IOI练习赛第1题）给定平面上n个点$P_i(i, y_i)$，P_i和P_j之间有边当且仅当$\forall i < k < j$，P_k在线段P_iP_j下方，求该图的最大独立集

- 20pts：$n \leq 19$
- 40pts：$n \leq 40$
- 70pts：$n \leq 200$
- 100pts：$n \leq 2000$

例：Mountains

（IOI练习赛第1题）给定平面上n个点$P_i(i, y_i)$，P_i和P_j之间有边当且仅当$\forall i < k < j$，P_k在线段P_iP_j下方，求该图的最大独立集

- 20pts：$n \leq 19$
- 40pts：$n \leq 40$
- 70pts：$n \leq 200$
- 100pts：$n \leq 2000$

例：Mountains

（IOI练习赛第1题）给定平面上n个点$P_i(i, y_i)$，P_i和P_j之间有边当且仅当$\forall i < k < j$，P_k在线段P_iP_j下方，求该图的最大独立集

- 20pts：$n \leq 19$
- 40pts：$n \leq 40$
- 70pts：$n \leq 200$
- 100pts：$n \leq 2000$

例：Mountains

（IOI练习赛第1题）给定平面上n个点$P_i(i, y_i)$，P_i和P_j之间有边当且仅当$\forall i < k < j$，P_k在线段P_iP_j下方，求该图的最大独立集

- 20pts：$n \leq 19$
- 40pts：$n \leq 40$
- 70pts：$n \leq 200$
- 100pts：$n \leq 2000$

例：Mountains

（IOI练习赛第1题）给定平面上n个点$P_i(i, y_i)$，P_i和P_j之间有边当且仅当$\forall i < k < j$，P_k在线段P_iP_j下方，求该图的最大独立集

- 20pts：$n \leq 19$
- 40pts：$n \leq 40$
- 70pts：$n \leq 200$
- 100pts：$n \leq 2000$

例：Mountains

- $O(2^n)$搜索：20pts
- $O(2^{\frac{n}{2}})$DP：40pts
- 加入对连通块分别处理的优化：70pts
- 把上述优化的DP输出一下搜出的状态，发现把编号最小的点及邻域删掉后，每个连通块编号是一段连续区间……
- 直接$O(n^2)$区间DP：100pts

考场上遇到不会做的题，不妨写点优秀的暴力，不仅能多拿不少分，还有机会发现性质哦！

例：Mountains

- $O(2^n)$搜索：20pts
- $O(2^{\frac{n}{2}})$DP：40pts
- 加入对连通块分别处理的优化：70pts
- 把上述优化的DP输出一下搜出的状态，发现把编号最小的点及邻域删掉后，每个连通块编号是一段连续区间……
- 直接$O(n^2)$区间DP：100pts

考场上遇到不会做的题，不妨写点优秀的暴力，不仅能多拿不少分，还有机会发现性质哦！

例：Mountains

- $O(2^n)$搜索：20pts
- $O(2^{\frac{n}{2}})$DP：40pts
- 加入对连通块分别处理的优化：70pts
- 把上述优化的DP输出一下搜出的状态，发现把编号最小的点及邻域删掉后，每个连通块编号是一段连续区间……
- 直接$O(n^2)$区间DP：100pts

考场上遇到不会做的题，不妨写点优秀的暴力，不仅能多拿不少分，还有机会发现性质哦！

例：Mountains

- $O(2^n)$搜索：20pts
- $O(2^{\frac{n}{2}})$DP：40pts
- 加入对连通块分别处理的优化：70pts
- 把上述优化的DP输出一下搜出的状态，发现把编号最小的点及邻域删掉后，每个连通块编号是一段连续区间……
- 直接$O(n^2)$区间DP：100pts

考场上遇到不会做的题，不妨写点优秀的暴力，不仅能多拿不少分，还有机会发现性质哦！

例：Mountains

- $O(2^n)$搜索：20pts
- $O(2^{\frac{n}{2}})$DP：40pts
- 加入对连通块分别处理的优化：70pts
- 把上述优化的DP输出一下搜出的状态，发现把编号最小的点及邻域删掉后，每个连通块编号是一段连续区间......
- 直接$O(n^2)$区间DP：100pts

考场上遇到不会做的题，不妨写点优秀的暴力，不仅能多拿不少分，还有机会发现性质哦！

近似及随机算法

近似算法、随机算法可能无法得到最优解，但有时可以得到很接近最优解的解

对于一些特殊的OI题、提交答案题、Codechef的Challenge等，近似及随机算法有广泛的应用

近似及随机算法

近似算法、随机算法可能无法得到最优解，但有时可以得到很接近最优解的解

对于一些特殊的OI题、提交答案题、Codechef的Challenge等，近似及随机算法有广泛的应用

贪心算法

贪心是最常用的骗分手段之一

按某种顺序贪心地将元素加入解中，得到较优解

但也有可能得到比最优解差得多的解

贪心算法

贪心是最常用的骗分手段之一
按某种顺序贪心地将元素加入解中，得到较优解
但也有可能得到比最优解差得多的解

贪心算法

贪心是最常用的骗分手段之一
按某种顺序贪心地将元素加入解中，得到较优解
但也有可能得到比最优解差得多的解

例：完全背包问题（骗分）

给定大小为n的物品集合中每个物品的重量w_i和价值v_i，以及一个容量C，每个物品可以选任意多次，满足总重量$\sum_{i \in S} w_i \leq C$，最大化总价值$V = \sum_{i \in S} v_i$

这个做法能得到近似解（至少最优解的$\frac{1}{2}$），甚至随机数据经常跑出最优解

PS：出完全背包的题数据不能纯随机

例：完全背包问题（骗分）

给定大小为n的物品集合中每个物品的重量w_i和价值v_i，以及一个容量C，每个物品可以选任意多次，满足总重量$\sum_{i\in S} w_i \le C$，最大化总价值$V=\sum_{i\in S} v_i$

这个做法能得到近似解（至少最优解的$\frac{1}{2}$），甚至随机数据经常跑出最优解

PS：出完全背包的题数据不能纯随机

例：完全背包问题（骗分）

给定大小为n的物品集合中每个物品的重量w_i和价值v_i，以及一个容量C，每个物品可以选任意多次，满足总重量$\sum_{i \in S} w_i \le C$，最大化总价值$V = \sum_{i \in S} v_i$

这个做法能得到近似解（至少最优解的$\frac{1}{2}$），甚至随机数据经常跑出最优解

PS：出完全背包的题数据不能纯随机

例：背包问题

如果0-1背包，或者数据较强，贪心效果就比较差了

一种比较靠谱的近似算法：

- 设定一个块大小B
- 将所有v_i修改为$v_i' = \lfloor \frac{v_i}{B} \rfloor$
- 做一个新的DP：$f(i,j)$为前i个物品取价值和至少为j的最少重量和，根据最优解得到方案

令$V = \max\{v_i\}$，则复杂度$O(n^2 \cdot \frac{V}{B})$

这个算法是可完全近似的，即：对于$\epsilon > 0$，令$V = \max\{v_i\}$，$B = \frac{\epsilon V}{n}$，就能以$O(\frac{n^3}{\epsilon})$的时间求出至少最优解的$1-\epsilon$倍的解

- 如果把重量而不是价值除以B上取整，效果如何?

例：背包问题

如果0-1背包，或者数据较强，贪心效果就比较差了
一种比较靠谱的近似算法：

- 设定一个块大小B
- 将所有v_i修改为$v_i' = \lfloor \frac{v_i}{B} \rfloor$
- 做一个新的DP：$f(i,j)$为前i个物品取价值和至少为j的最少重量和，根据最优解得到方案

令$V = \max\{v_i\}$，则复杂度$O(n^2 \cdot \frac{V}{B})$
这个算法是可完全近似的，即：对于$\epsilon > 0$，
令$V = \max\{v_i\}$，$B = \frac{\epsilon V}{n}$，就能以$O(\frac{n^3}{\epsilon})$的时间求出至少最优解的$1 - \epsilon$倍的解

- 如果把重量而不是价值除以B上取整，效果如何？

例：背包问题

如果0-1背包，或者数据较强，贪心效果就比较差了
一种比较靠谱的近似算法：

- 设定一个块大小B
- 将所有v_i修改为$v_i' = \lfloor \frac{v_i}{B} \rfloor$
- 做一个新的DP：$f(i,j)$为前i个物品取价值和至少为j的最少重量和，根据最优解得到方案

令$V = \max\{v_i\}$，则复杂度$O(n^2 \cdot \frac{V}{B})$
这个算法是可完全近似的，即：对于$\epsilon > 0$，
令$V = \max\{v_i\}$，$B = \frac{\epsilon V}{n}$，就能以$O(\frac{n^3}{\epsilon})$的时间求出至少最优解的$1-\epsilon$倍的解

- 如果把重量而不是价值除以B上取整，效果如何？

例：背包问题

如果0-1背包，或者数据较强，贪心效果就比较差了
一种比较靠谱的近似算法：

- 设定一个块大小B
- 将所有v_i修改为$v_i' = \lfloor \frac{v_i}{B} \rfloor$
- 做一个新的DP：$f(i,j)$为前i个物品取价值和至少为j的最少重量和，根据最优解得到方案

令$V = \max\{v_i\}$，则复杂度$O(n^2 \cdot \frac{V}{B})$
这个算法是可完全近似的，即：对于$\epsilon > 0$，
令$V = \max\{v_i\}$，$B = \frac{\epsilon V}{n}$，就能以$O(\frac{n^3}{\epsilon})$的时间求出至少最优解的$1-\epsilon$倍的解

- 如果把重量而不是价值除以B上取整，效果如何?

例：背包问题

如果0-1背包，或者数据较强，贪心效果就比较差了
一种比较靠谱的近似算法：

- 设定一个块大小B
- 将所有v_i修改为$v_i' = \lfloor \frac{v_i}{B} \rfloor$
- 做一个新的DP：$f(i,j)$为前i个物品取价值和至少为j的最少重量和，根据最优解得到方案

令$V = \max\{v_i\}$，则复杂度$O(n^2 \cdot \frac{V}{B})$
这个算法是可完全近似的，即：对于$\epsilon > 0$，
令$V = \max\{v_i\}$，$B = \frac{\epsilon V}{n}$，就能以$O(\frac{n^3}{\epsilon})$的时间求出至少最优解的$1-\epsilon$倍的解

- 如果把重量而不是价值除以B上取整，效果如何?

例：背包问题

如果0-1背包，或者数据较强，贪心效果就比较差了
一种比较靠谱的近似算法：

- 设定一个块大小B
- 将所有v_i修改为$v_i' = \lfloor \frac{v_i}{B} \rfloor$
- 做一个新的DP：$f(i,j)$为前i个物品取价值和至少为j的最少重量和，根据最优解得到方案

令$V = \max\{v_i\}$，则复杂度$O(n^2 \cdot \frac{V}{B})$

这个算法是可完全近似的，即：对于$\epsilon > 0$，
令$V = \max\{v_i\}$，$B = \frac{\epsilon V}{n}$，就能以$O(\frac{n^3}{\epsilon})$的时间求出至少最优解的$1-\epsilon$倍的解

- 如果把重量而不是价值除以B上取整，效果如何？

例：背包问题

如果0-1背包，或者数据较强，贪心效果就比较差了
一种比较靠谱的近似算法：

- 设定一个块大小B
- 将所有v_i修改为$v_i' = \lfloor \frac{v_i}{B} \rfloor$
- 做一个新的DP：$f(i,j)$为前i个物品取价值和至少为j的最少重量和，根据最优解得到方案

令$V = \max\{v_i\}$，则复杂度$O(n^2 \cdot \frac{V}{B})$
这个算法是可完全近似的，即：对于$\epsilon > 0$，
令$V = \max\{v_i\}$，$B = \frac{\epsilon V}{n}$，就能以$O(\frac{n^3}{\epsilon})$的时间求出至少最优解的$1 - \epsilon$倍的解

- 如果把重量而不是价值除以B上取整，效果如何?

例：背包问题

如果0-1背包，或者数据较强，贪心效果就比较差了
一种比较靠谱的近似算法：

- 设定一个块大小B
- 将所有v_i修改为$v_i' = \lfloor \frac{v_i}{B} \rfloor$
- 做一个新的DP：$f(i,j)$为前i个物品取价值和至少为j的最少重量和，根据最优解得到方案

令$V = \max\{v_i\}$，则复杂度$O(n^2 \cdot \frac{V}{B})$
这个算法是可完全近似的，即：对于$\epsilon > 0$，
令$V = \max\{v_i\}$，$B = \frac{\epsilon V}{n}$，就能以$O(\frac{n^3}{\epsilon})$的时间求出至少最优解的$1 - \epsilon$倍的解

- 如果把重量而不是价值除以B上取整，效果如何?

随机算法

当解空间中最优解/较优解分布密集时，可以只搜索解空间的一部分

- 朴素的随机法：仿照贪心法，以随机序将元素加入解中，多次随机产生解并取最优
- 有些问题可以对解的一部分进行随机，用确定性算法求解剩下一部分

随机算法

当解空间中最优解/较优解分布密集时，可以只搜索解空间的一部分

- 朴素的随机法：仿照贪心法，以随机序将元素加入解中，多次随机产生解并取最优
- 有些问题可以对解的一部分进行随机，用确定性算法求解剩下一部分

随机算法

当解空间中最优解/较优解分布密集时，可以只搜索解空间的一部分

- 朴素的随机法：仿照贪心法，以随机序将元素加入解中，多次随机产生解并取最优
- 有些问题可以对解的一部分进行随机，用确定性算法求解剩下一部分

例：独立集问题（骗分）

给定无向图$G=(V,E)$，求一个$S\subseteq V$，满足S中的点两两不相邻，最大化$|S|$

当图非常大的时候，搜索和DP都跑不出来，不过朴素的随机法可以得到一个比较大的独立集

- 随机排列点集$v_1,v_2,\cdots,v_n$，初始$I=\varnothing$
- 按$i=1,2,\cdots,n$顺序，如果$I+\{v_i\}$是独立集则将v_i加入I
- 最后得到极大独立集I
- 多次随机取最优解
- 另外，第二步如果$I+\{v_i\}$加上去掉邻域的剩余点数不大于当前最优解则不加入I

不妨试试看？

例：独立集问题（骗分）

给定无向图$G=(V,E)$，求一个$S\subseteq V$，满足S中的点两两不相邻，最大化$|S|$

当图非常大的时候，搜索和DP都跑不出来，不过朴素的随机法可以得到一个比较大的独立集

- 随机排列点集$v_1, v_2, \cdots, v_n$，初始$I=\varnothing$
- 按$i=1,2,\cdots,n$顺序，如果$I+\{v_i\}$是独立集则将v_i加入I
- 最后得到极大独立集I
- 多次随机取最优解
- 另外，第二步如果$I+\{v_i\}$加上去掉邻域的剩余点数不大于当前最优解则不加入I

不妨试试看？

例：独立集问题（骗分）

给定无向图$G=(V,E)$，求一个$S\subseteq V$，满足S中的点两两不相邻，最大化$|S|$

当图非常大的时候，搜索和DP都跑不出来，不过朴素的随机法可以得到一个比较大的独立集

- 随机排列点集$v_1,v_2,\cdots,v_n$，初始$I=\varnothing$
- 按$i=1,2,\cdots,n$顺序，如果$I+\{v_i\}$是独立集则将v_i加入I
- 最后得到极大独立集I
- 多次随机取最优解
- 另外，第二步如果$I+\{v_i\}$加上去掉邻域的剩余点数不大于当前最优解则不加入I

不妨试试看?

例：独立集问题（骗分）

给定无向图$G=(V,E)$，求一个$S\subseteq V$，满足S中的点两两不相邻，最大化$|S|$

当图非常大的时候，搜索和DP都跑不出来，不过朴素的随机法可以得到一个比较大的独立集

- 随机排列点集$v_1,v_2,\cdots,v_n$，初始$I=\varnothing$
- 按$i=1,2,\cdots,n$顺序，如果$I+\{v_i\}$是独立集则将v_i加入I
- 最后得到极大独立集I
- 多次随机取最优解
- 另外，第二步如果$I+\{v_i\}$加上去掉邻域的剩余点数不大于当前最优解则不加入I

不妨试试看？

例：独立集问题（骗分）

给定无向图$G = (V, E)$，求一个$S \subseteq V$，满足S中的点两两不相邻，最大化$|S|$

当图非常大的时候，搜索和DP都跑不出来，不过朴素的随机法可以得到一个比较大的独立集

- 随机排列点集$v_1, v_2, \cdots, v_n$，初始$I = \varnothing$
- 按$i = 1, 2, \cdots, n$顺序，如果$I + \{v_i\}$是独立集则将v_i加入I
- 最后得到极大独立集I
- 多次随机取最优解
- 另外，第二步如果$I + \{v_i\}$加上去掉邻域的剩余点数不大于当前最优解则不加入I

不妨试试看?

例：独立集问题（骗分）

给定无向图$G=(V,E)$，求一个$S\subseteq V$，满足S中的点两两不相邻，最大化$|S|$

当图非常大的时候，搜索和DP都跑不出来，不过朴素的随机法可以得到一个比较大的独立集

- 随机排列点集$v_1,v_2,\cdots,v_n$，初始$I=\varnothing$
- 按$i=1,2,\cdots,n$顺序，如果$I+\{v_i\}$是独立集则将v_i加入I
- 最后得到极大独立集I
- 多次随机取最优解
- 另外，第二步如果$I+\{v_i\}$加上去掉邻域的剩余点数不大于当前最优解则不加入I

不妨试试看?

例：独立集问题（骗分）

给定无向图$G=(V,E)$，求一个$S\subseteq V$，满足S中的点两两不相邻，最大化$|S|$

当图非常大的时候，搜索和DP都跑不出来，不过朴素的随机法可以得到一个比较大的独立集

- 随机排列点集$v_1,v_2,\cdots,v_n$，初始$I=\varnothing$
- 按$i=1,2,\cdots,n$顺序，如果$I+\{v_i\}$是独立集则将v_i加入I
- 最后得到极大独立集I
- 多次随机取最优解
- 另外，第二步如果$I+\{v_i\}$加上去掉邻域的剩余点数不大于当前最优解则不加入I

不妨试试看？

例：独立集问题（骗分）

给定无向图$G=(V,E)$，求一个$S \subseteq V$，满足S中的点两两不相邻，最大化$|S|$

当图非常大的时候，搜索和DP都跑不出来，不过朴素的随机法可以得到一个比较大的独立集

- 随机排列点集$v_1, v_2, \cdots, v_n$，初始$I=\varnothing$
- 按$i=1,2,\cdots,n$顺序，如果$I+\{v_i\}$是独立集则将v_i加入I
- 最后得到极大独立集I
- 多次随机取最优解
- 另外，第二步如果$I+\{v_i\}$加上去掉邻域的剩余点数不大于当前最优解则不加入I

不妨试试看？

例：独立集问题（骗分）

事实证明随机数据经常能找到接近最大的独立集，有时甚至只比最大独立集小1

WC2013第3题（小Q运动季）测试点10的最优解为34，这个算法可以跑出33的解，得到9分，如果考场上时间比较紧，写这个算法是一种不错的考场策略

例：独立集问题（骗分）

事实证明随机数据经常能找到接近最大的独立集，有时甚至只比最大独立集小1

WC2013第3题（小Q运动季）测试点10的最优解为34，这个算法可以跑出33的解，得到9分，如果考场上时间比较紧，写这个算法是一种不错的考场策略

例：THUSC2017 D1T1 简化版

给定无向图$G=(V,E)$，每个结点v有一个颜色$c_v\in\mathbf{N}^*$和一个正整数代价$w_v\in\mathbf{N}^*$

求G的一个包含至少k种颜色的连通子图G'，使得代价和$W=\sum_{v\in V(G')}w_v$最小

k的值比较小

例：THUSC2017 D1T1 简化版

给定无向图$G=(V,E)$，每个结点v有一个颜色$c_v\in\mathbf{N}^*$和一个正整数代价$w_v\in\mathbf{N}^*$

求G的一个包含至少k种颜色的连通子图G'，使得代价和$W=\sum_{v\in V(G')}w_v$最小

k的值比较小

例：THUSC2017 D1T1 简化版

给定无向图$G=(V,E)$，每个结点v有一个颜色$c_v\in\mathbf{N}^*$和一个正整数代价$w_v\in\mathbf{N}^*$

求G的一个包含至少k种颜色的连通子图G'，使得代价和$W=\sum_{v\in V(G')}w_v$最小

k的值比较小

例：THUSC2017 D1T1 简化版

直接多次随机连通块?

一种随机化算法：

- 先把图上的原有颜色随机分成k组，记为$1, 2, \cdots, k$
- 然后记$f(i, S)$为包含点i以及集合S内的所有颜色组的最小代价和（$S \subseteq \{1, 2, \cdots, k\}$），则

$$f(i,S) = \min \begin{cases} \min_{T \subsetneq S, T \neq \varnothing}\{f(i,T) + f(i,S-T)\}, \\ w_i + \min_{(i,j)\in E}\{f(j,S)\} \end{cases}$$

  第二类转移用最短路
- 多次随机取最优解

每次有$O(\frac{k!}{k^k})$的概率取得最优解

例：THUSC2017 D1T1 简化版

直接多次随机连通块?
一种随机化算法：

- 先把图上的原有颜色随机分成k组，记为$1, 2, \cdots, k$
- 然后记$f(i, S)$为包含点i以及集合S内的所有颜色组的最小代价和（$S \subseteq \{1, 2, \cdots, k\}$），则

$$f(i,S) = \min \begin{cases} \min_{T \subsetneq S, T \neq \varnothing}\{f(i,T) + f(i, S-T)\}, \\ w_i + \min_{(i,j)\in E}\{f(j,S)\} \end{cases}$$

  第二类转移用最短路
- 多次随机取最优解

每次有$O(\frac{k!}{k^k})$的概率取得最优解

例：THUSC2017 D1T1 简化版

直接多次随机连通块?
一种随机化算法：

- 先把图上的原有颜色随机分成k组，记为$1,2,\cdots,k$
- 然后记$f(i,S)$为包含点i以及集合S内的所有颜色组的最小代价和（$S\subseteq\{1,2,\cdots,k\}$），则

$$f(i,S)=\min\begin{cases}\min_{T\subsetneq S,T\neq\varnothing}\{f(i,T)+f(i,S-T)\},\\ w_i+\min_{(i,j)\in E}\{f(j,S)\}\end{cases}$$

第二类转移用最短路

- 多次随机取最优解

每次有$O(\frac{k!}{k^k})$的概率取得最优解

例：THUSC2017 D1T1 简化版

直接多次随机连通块?
一种随机化算法：

- 先把图上的原有颜色随机分成k组，记为$1,2,\cdots,k$
- 然后记$f(i,S)$为包含点i以及集合S内的所有颜色组的最小代价和（$S\subseteq\{1,2,\cdots,k\}$），则

$$f(i,S)=\min\begin{cases}\min_{T\subsetneq S,T\neq\varnothing}\{f(i,T)+f(i,S-T)\},\\ w_i+\min_{(i,j)\in E}\{f(j,S)\}\end{cases}$$

  第二类转移用最短路
- 多次随机取最优解

每次有$O(\frac{k!}{k^k})$的概率取得最优解

例：THUSC2017 D1T1 简化版

直接多次随机连通块?
一种随机化算法：

- 先把图上的原有颜色随机分成k组，记为$1, 2, \cdots, k$
- 然后记$f(i, S)$为包含点i以及集合S内的所有颜色组的最小代价和（$S \subseteq \{1, 2, \cdots, k\}$），则

$$f(i, S) = \min \begin{cases} \min_{T \subsetneq S, T \neq \varnothing}\{f(i, T) + f(i, S - T)\}, \\ w_i + \min_{(i,j) \in E}\{f(j, S)\} \end{cases}$$

  第二类转移用最短路

- 多次随机取最优解

每次有$O(\frac{k!}{k^k})$的概率取得最优解

局部搜索算法

局部搜索是一种应用广泛的近似算法

本质是在搜索过程中沿局部最优的方向搜索

- 爬山算法
- 模拟退火算法
- 遗传算法
- 蚁群算法
- ……

局部搜索算法

局部搜索是一种应用广泛的近似算法
本质是在搜索过程中沿局部最优的方向搜索

- 爬山算法
- 模拟退火算法
- 遗传算法
- 蚁群算法
- ……

局部搜索算法

局部搜索是一种应用广泛的近似算法
本质是在搜索过程中沿局部最优的方向搜索

- 爬山算法
- 模拟退火算法
- 遗传算法
- 蚁群算法
- ……

局部搜索算法

局部搜索是一种应用广泛的近似算法
本质是在搜索过程中沿局部最优的方向搜索

- 爬山算法
- 模拟退火算法
- 遗传算法
- 蚁群算法
- ……

局部搜索算法

局部搜索是一种应用广泛的近似算法
本质是在搜索过程中沿局部最优的方向搜索

- 爬山算法
- 模拟退火算法
- 遗传算法
- 蚁群算法
- ……

局部搜索算法

局部搜索是一种应用广泛的近似算法
本质是在搜索过程中沿局部最优的方向搜索

- 爬山算法
- 模拟退火算法
- 遗传算法
- 蚁群算法
- ……

局部搜索算法

局部搜索是一种应用广泛的近似算法
本质是在搜索过程中沿局部最优的方向搜索

- 爬山算法
- 模拟退火算法
- 遗传算法
- 蚁群算法
- ……

爬山算法

最优化问题可以认为是在解空间中寻找一个解x，最小化目标函数$f(x)$

- 随机选取初始解x_0
- 每次从x_0的邻域中选取一个新解$x \in N(x_0)$，满足$f(x) < f(x_0)$，然后更新x_0为x，重复直至$N(x_0)$中不存在满足$f(x) < f(x_0)$的x
- 多次随机x_0取最优解

爬山算法

最优化问题可以认为是在解空间中寻找一个解x，最小化目标函数$f(x)$

- 随机选取初始解x_0
- 每次从x_0的邻域中选取一个新解$x \in N(x_0)$，满足$f(x) < f(x_0)$，然后更新x_0为x，重复直至$N(x_0)$中不存在满足$f(x) < f(x_0)$的x
- 多次随机x_0取最优解

爬山算法

最优化问题可以认为是在解空间中寻找一个解x，最小化目标函数$f(x)$

- 随机选取初始解x_0
- 每次从x_0的邻域中选取一个新解$x \in N(x_0)$，满足$f(x) < f(x_0)$，然后更新x_0为x，重复直至$N(x_0)$中不存在满足$f(x) < f(x_0)$的x
- 多次随机x_0取最优解

爬山算法

最优化问题可以认为是在解空间中寻找一个解x，最小化目标函数$f(x)$

- 随机选取初始解x_0
- 每次从x_0的邻域中选取一个新解$x \in N(x_0)$，满足$f(x) < f(x_0)$，然后更新x_0为x，重复直至$N(x_0)$中不存在满足$f(x) < f(x_0)$的x
- 多次随机x_0取最优解

爬山算法

最优化问题可以认为是在解空间中寻找一个解x，最小化目标函数$f(x)$

- 随机选取初始解x_0
- 每次从x_0的邻域中选取一个新解$x \in N(x_0)$，满足$f(x) < f(x_0)$，然后更新x_0为x，重复直至$N(x_0)$中不存在满足$f(x) < f(x_0)$的x
- 多次随机x_0取最优解

模拟退火算法

模拟固体的退火过程改进局部搜索

在模拟退火算法中，比当前解劣的新解也有一定概率被接受

- 随机选取初始解x_0，设定初始温度$t = t_0$
- 每次从x_0的邻域中选取一个新解$x \in N(x_0)$，有$P(x_0, x)$的概率更新x_0为x，其中

$$P(x_0, x) = \begin{cases} 1, & f(x) < f(x_0), \\ e^{\frac{f(x_0)-f(x)}{t}}, & f(x) \geq f(x_0) \end{cases}$$

然后降低t的值，迭代该过程直至最优解稳定

- 多次随机x_0取最优解

模拟退火的参数对解的影响很大

模拟退火算法

模拟固体的退火过程改进局部搜索

在模拟退火算法中，比当前解劣的新解也有一定概率被接受

- 随机选取初始解x_0，设定初始温度$t = t_0$
- 每次从x_0的邻域中选取一个新解$x \in N(x_0)$，有$P(x_0, x)$的概率更新x_0为x，其中

$$P(x_0, x) = \begin{cases} 1, & f(x) < f(x_0), \\ e^{\frac{f(x_0) - f(x)}{t}}, & f(x) \geq f(x_0) \end{cases}$$

  然后降低t的值，迭代该过程直至最优解稳定
- 多次随机x_0取最优解

模拟退火的参数对解的影响很大

模拟退火算法

模拟固体的退火过程改进局部搜索
在模拟退火算法中，比当前解劣的新解也有一定概率被接受

- 随机选取初始解x_0，设定初始温度$t = t_0$
- 每次从x_0的邻域中选取一个新解$x \in N(x_0)$，有$P(x_0, x)$的概率更新x_0为x，其中

$$P(x_0, x) = \begin{cases} 1, & f(x) < f(x_0), \\ e^{\frac{f(x_0)-f(x)}{t}}, & f(x) \geq f(x_0) \end{cases}$$

  然后降低t的值，迭代该过程直至最优解稳定
- 多次随机x_0取最优解

模拟退火的参数对解的影响很大

模拟退火算法

模拟固体的退火过程改进局部搜索
在模拟退火算法中，比当前解劣的新解也有一定概率被接受

- 随机选取初始解x_0，设定初始温度$t = t_0$
- 每次从x_0的邻域中选取一个新解$x \in N(x_0)$，有$P(x_0, x)$的概率更新x_0为x，其中

$$P(x_0, x) = \begin{cases} 1, & f(x) < f(x_0), \\ e^{\frac{f(x_0)-f(x)}{t}}, & f(x) \geq f(x_0) \end{cases}$$

然后降低t的值，迭代该过程直至最优解稳定

- 多次随机x_0取最优解

模拟退火的参数对解的影响很大

模拟退火算法

模拟固体的退火过程改进局部搜索
在模拟退火算法中，比当前解劣的新解也有一定概率被接受

- 随机选取初始解x_0，设定初始温度$t = t_0$
- 每次从x_0的邻域中选取一个新解$x \in N(x_0)$，有$P(x_0, x)$的概率更新x_0为x，其中

$$P(x_0, x) = \begin{cases} 1, & f(x) < f(x_0), \\ e^{\frac{f(x_0)-f(x)}{t}}, & f(x) \geq f(x_0) \end{cases}$$

  然后降低t的值，迭代该过程直至最优解稳定
- 多次随机x_0取最优解

模拟退火的参数对解的影响很大

模拟退火算法

模拟固体的退火过程改进局部搜索

在模拟退火算法中，比当前解劣的新解也有一定概率被接受

- 随机选取初始解x_0，设定初始温度$t = t_0$
- 每次从x_0的邻域中选取一个新解$x \in N(x_0)$，有$P(x_0, x)$的概率更新x_0为x，其中

$$P(x_0, x) = \begin{cases} 1, & f(x) < f(x_0), \\ e^{\frac{f(x_0)-f(x)}{t}}, & f(x) \geq f(x_0) \end{cases}$$

  然后降低t的值，迭代该过程直至最优解稳定
- 多次随机x_0取最优解

模拟退火的参数对解的影响很大

遗传算法

模拟达尔文生物进化论的自然选择的一种局部搜索算法

- 个体：问题的一个解
- 染色体：解的编码
- 基因：编码的元素
- 群体：被选定的一组解
- 环境：适应函数
- 适应度：适应函数的值

遗传算法

模拟达尔文生物进化论的自然选择的一种局部搜索算法

- 个体：问题的一个解
- 染色体：解的编码
- 基因：编码的元素
- 群体：被选定的一组解
- 环境：适应函数
- 适应度：适应函数的值

遗传算法

模拟达尔文生物进化论的自然选择的一种局部搜索算法

- 个体：问题的一个解
- 染色体：解的编码
- 基因：编码的元素
- 群体：被选定的一组解
- 环境：适应函数
- 适应度：适应函数的值

遗传算法

模拟达尔文生物进化论的自然选择的一种局部搜索算法

- 个体：问题的一个解
- 染色体：解的编码
- 基因：编码的元素
- 群体：被选定的一组解
- 环境：适应函数
- 适应度：适应函数的值

遗传算法

模拟达尔文生物进化论的自然选择的一种局部搜索算法

- 个体：问题的一个解
- 染色体：解的编码
- 基因：编码的元素
- 群体：被选定的一组解
- 环境：适应函数
- 适应度：适应函数的值

遗传算法

模拟达尔文生物进化论的自然选择的一种局部搜索算法

- 个体：问题的一个解
- 染色体：解的编码
- 基因：编码的元素
- 群体：被选定的一组解
- 环境：适应函数
- 适应度：适应函数的值

遗传算法

模拟达尔文生物进化论的自然选择的一种局部搜索算法

- 个体：问题的一个解
- 染色体：解的编码
- 基因：编码的元素
- 群体：被选定的一组解
- 环境：适应函数
- 适应度：适应函数的值

遗传算法

主要操作有三种：

- 选择：根据适应函数值选择个体
- 交叉：以一定的方式由双亲产生后代
- 变异：染色体的某些基因发生变化

遗传算法

主要操作有三种：

- 选择：根据适应函数值选择个体
- 交叉：以一定的方式由双亲产生后代
- 变异：染色体的某些基因发生变化

遗传算法

主要操作有三种：

- 选择：根据适应函数值选择个体
- 交叉：以一定的方式由双亲产生后代
- 变异：染色体的某些基因发生变化

遗传算法

主要操作有三种：

- 选择：根据适应函数值选择个体
- 交叉：以一定的方式由双亲产生后代
- 变异：染色体的某些基因发生变化

遗传算法

算法过程：

- 设定群体规模N，交叉概率p_c和变异概率p_m
- 随机生成N个染色体作为初始群体
- 根据交叉概率p_c选择染色体进行交叉，子代及未进行交叉的染色体进入新群体
- 根据变异概率p_m从新群体中选择染色体进行变异取代原染色体
- 当迭代次数达到上限时结束，输出适应度$F(x_i)$最大的染色体x_i
- 否则，根据所有染色体x_i的适应度$F(x_i)$保留N个染色体，回到第三步

遗传算法

算法过程：

- 设定群体规模N，交叉概率p_c和变异概率p_m
- 随机生成N个染色体作为初始群体
- 根据交叉概率p_c选择染色体进行交叉，子代及未进行交叉的染色体进入新群体
- 根据变异概率p_m从新群体中选择染色体进行变异取代原染色体
- 当迭代次数达到上限时结束，输出适应度$F(x_i)$最大的染色体x_i
- 否则，根据所有染色体x_i的适应度$F(x_i)$保留N个染色体，回到第三步

遗传算法

算法过程：

- 设定群体规模N，交叉概率p_c和变异概率p_m
- 随机生成N个染色体作为初始群体
- 根据交叉概率p_c选择染色体进行交叉，子代及未进行交叉的染色体进入新群体
- 根据变异概率p_m从新群体中选择染色体进行变异取代原染色体
- 当迭代次数达到上限时结束，输出适应度$F(x_i)$最大的染色体x_i
- 否则，根据所有染色体x_i的适应度$F(x_i)$保留N个染色体，回到第三步

遗传算法

算法过程：

- 设定群体规模N，交叉概率p_c和变异概率p_m
- 随机生成N个染色体作为初始群体
- 根据交叉概率p_c选择染色体进行交叉，子代及未进行交叉的染色体进入新群体
- 根据变异概率p_m从新群体中选择染色体进行变异取代原染色体
- 当迭代次数达到上限时结束，输出适应度$F(x_i)$最大的染色体x_i
- 否则，根据所有染色体x_i的适应度$F(x_i)$保留N个染色体，回到第三步

遗传算法

算法过程：

- 设定群体规模N，交叉概率p_c和变异概率p_m
- 随机生成N个染色体作为初始群体
- 根据交叉概率p_c选择染色体进行交叉，子代及未进行交叉的染色体进入新群体
- 根据变异概率p_m从新群体中选择染色体进行变异取代原染色体
- 当迭代次数达到上限时结束，输出适应度$F(x_i)$最大的染色体x_i
- 否则，根据所有染色体x_i的适应度$F(x_i)$保留N个染色体，回到第三步

遗传算法

算法过程：

- 设定群体规模N，交叉概率p_c和变异概率p_m
- 随机生成N个染色体作为初始群体
- 根据交叉概率p_c选择染色体进行交叉，子代及未进行交叉的染色体进入新群体
- 根据变异概率p_m从新群体中选择染色体进行变异取代原染色体
- 当迭代次数达到上限时结束，输出适应度$F(x_i)$最大的染色体x_i
- 否则，根据所有染色体x_i的适应度$F(x_i)$保留N个染色体，回到第三步

遗传算法

算法过程：

- 设定群体规模N，交叉概率p_c和变异概率p_m
- 随机生成N个染色体作为初始群体
- 根据交叉概率p_c选择染色体进行交叉，子代及未进行交叉的染色体进入新群体
- 根据变异概率p_m从新群体中选择染色体进行变异取代原染色体
- 当迭代次数达到上限时结束，输出适应度$F(x_i)$最大的染色体x_i
- 否则，根据所有染色体x_i的适应度$F(x_i)$保留N个染色体，回到第三步

遗传算法

交叉：由两个父代染色体（双亲）产生两个具有双亲的部分基因的新的染色体

二进制编码举例：

- 交叉前：

$$a_1, a_2, \cdots, a_i, a_{i+1}, \cdots, a_n$$
$$b_1, b_2, \cdots, b_i, b_{i+1}, \cdots, b_n$$

- 交叉后：

$$a_1, a_2, \cdots, a_i, b_{i+1}, \cdots, b_n$$
$$b_1, b_2, \cdots, b_i, a_{i+1}, \cdots, a_n$$

遗传算法

交叉：由两个父代染色体（双亲）产生两个具有双亲的部分基因的新的染色体

二进制编码举例：

- 交叉前：

$$a_1, a_2, \cdots, a_i, a_{i+1}, \cdots, a_n$$
$$b_1, b_2, \cdots, b_i, b_{i+1}, \cdots, b_n$$

- 交叉后：

$$a_1, a_2, \cdots, a_i, b_{i+1}, \cdots, b_n$$
$$b_1, b_2, \cdots, b_i, a_{i+1}, \cdots, a_n$$

遗传算法

交叉：由两个父代染色体（双亲）产生两个具有双亲的部分基因的新的染色体

二进制编码举例：

- 交叉前：

$$a_1, a_2, \cdots, a_i, a_{i+1}, \cdots, a_n$$
$$b_1, b_2, \cdots, b_i, b_{i+1}, \cdots, b_n$$

- 交叉后：

$$a_1, a_2, \cdots, a_i, b_{i+1}, \cdots, b_n$$
$$b_1, b_2, \cdots, b_i, a_{i+1}, \cdots, a_n$$

遗传算法

变异：染色体的某一个基因发射改变

二进制编码举例：

$11001 \rightarrow 11101$（变异发生在第3位）

遗传算法

变异：染色体的某一个基因发射改变
二进制编码举例：
11001 $\rightarrow$ 11101（变异发生在第3位）

遗传算法

变异：染色体的某一个基因发射改变
二进制编码举例：
11001 $\rightarrow$ 11101（变异发生在第3位）

遗传算法

选择：根据适应值保留个体

- 每次随机选取一个染色体，染色体x_i被选中的概率为$\frac{F(x_i)}{\sum_i F(x_i)}$，选取$N$个

遗传算法

选择：根据适应值保留个体

- 每次随机选取一个染色体，染色体x_i被选中的概率为$\frac{F(x_i)}{\sum_i F(x_i)}$，选取$N$个

遗传算法

选择：根据适应值保留个体

- 每次随机选取一个染色体，染色体x_i被选中的概率为$\frac{F(x_i)}{\sum_i F(x_i)}$，选取$N$个

遗传算法

定理：若遗传算法每次保留目前最优解，则当进化代数$T \to +\infty$时，遗传算法得到最优解的概率为1

蚁群算法

模仿蚂蚁群体的寻找路径行为的优化算法

蚂蚁会在经过的路径上留下信息素，蚁群会沿着信息素较高的路径行走

较短的路径上信息素较多，选择较短路径的蚂蚁数目也会逐渐增加，最后集中到最优的路径上

蚁群算法

模仿蚂蚁群体的寻找路径行为的优化算法

蚂蚁会在经过的路径上留下信息素，蚁群会沿着信息素较高的路径行走

较短的路径上信息素较多，选择较短路径的蚂蚁数目也会逐渐增加，最后集中到最优的路径上

蚁群算法

模仿蚂蚁群体的寻找路径行为的优化算法

蚂蚁会在经过的路径上留下信息素，蚁群会沿着信息素较高的路径行走

较短的路径上信息素较多，选择较短路径的蚂蚁数目也会逐渐增加，最后集中到最优的路径上

蚁群算法

- 生成N只蚂蚁，初始化所有路径信息素强度相等$\tau_{i,j} = C$（C为常数）
- 对每只蚂蚁，根据系统的信息素，以一定概率选择下一个结点，不断重复直到确定路径
- 对每只蚂蚁计算路径长度，更新系统的信息素
- 重复迭代上述过程，直到迭代次数达到上限结束

蚁群算法

- 生成N只蚂蚁，初始化所有路径信息素强度相等$\tau_{i,j} = C$（C为常数）
- 对每只蚂蚁，根据系统的信息素，以一定概率选择下一个结点，不断重复直到确定路径
- 对每只蚂蚁计算路径长度，更新系统的信息素
- 重复迭代上述过程，直到迭代次数达到上限结束

蚁群算法

- 生成N只蚂蚁，初始化所有路径信息素强度相等$\tau_{i,j} = C$（C为常数）
- 对每只蚂蚁，根据系统的信息素，以一定概率选择下一个结点，不断重复直到确定路径
- 对每只蚂蚁计算路径长度，更新系统的信息素
- 重复迭代上述过程，直到迭代次数达到上限结束

蚁群算法

- 生成N只蚂蚁，初始化所有路径信息素强度相等$\tau_{i,j} = C$（C为常数）
- 对每只蚂蚁，根据系统的信息素，以一定概率选择下一个结点，不断重复直到确定路径
- 对每只蚂蚁计算路径长度，更新系统的信息素
- 重复迭代上述过程，直到迭代次数达到上限结束

例：旅行商问题

给定n阶无向（即对称）或有向（即非对称）完全图$G=(V,E)$和权函数$w:E\to \mathbf{N}^*$，求G的一条经过每个结点恰好一次的回路C，使得$\sum_{e\in E(C)} w(e)$最小

- 朴素搜索：$O(n!)$
- 动态规划：$O(2^n n^2)$

例：旅行商问题

给定n阶无向（即对称）或有向（即非对称）完全图$G=(V,E)$和权函数$w: E \to \mathbf{N}^*$，求G的一条经过每个结点恰好一次的回路C，使得$\sum_{e \in E(C)} w(e)$最小

- 朴素搜索：$O(n!)$
- 动态规划：$O(2^n n^2)$

例：旅行商问题

给定n阶无向（即对称）或有向（即非对称）完全图$G=(V,E)$和权函数$w: E \to \mathbf{N}^*$，求G的一条经过每个结点恰好一次的回路C，使得$\sum_{e \in E(C)} w(e)$最小

- 朴素搜索：$O(n!)$
- 动态规划：$O(2^n n^2)$

TSPLIB

TSPLIB是一套公开的TSP问题库，包含了大量的TSP实例及其目前最优解

为了方便对比，以下测试数据均为TSPLIB中的数据

局限性：数据为平面上点的距离，和真正的随机图有一定差异

TSPLIB

TSPLIB是一套公开的TSP问题库，包含了大量的TSP实例及其目前最优解

为了方便对比，以下测试数据均为TSPLIB中的数据

局限性：数据为平面上点的距离，和真正的随机图有一定差异

TSPLIB

TSPLIB是一套公开的TSP问题库，包含了大量的TSP实例及其目前最优解

为了方便对比，以下测试数据均为TSPLIB中的数据

局限性：数据为平面上点的距离，和真正的随机图有一定差异

例：旅行商问题

让我们尝试一下爬山算法

- 随机生成初始回路$(p_0, p_1, \cdots, p_{n-1})$，规定$p_i = p_{i \bmod n}$
- 找一对i, j，如果$w(p_{i-1}, p_j) + w(p_i, p_{j+1}) < w(p_{i-1}, p_i) + w(p_j, p_{j+1})$则反转$p_i \cdots p_j$一段
- 当没有满足条件的i, j时结束
- 多次重复以上过程取最优

例：旅行商问题

让我们尝试一下爬山算法

- 随机生成初始回路$(p_0, p_1, \cdots, p_{n-1})$，规定$p_i = p_{i \bmod n}$
- 找一对i, j，如果$w(p_{i-1}, p_j) + w(p_i, p_{j+1}) < w(p_{i-1}, p_i) + w(p_j, p_{j+1})$则反转$p_i \cdots p_j$一段
- 当没有满足条件的i, j时结束
- 多次重复以上过程取最优

例：旅行商问题

让我们尝试一下爬山算法

- 随机生成初始回路$(p_0, p_1, \cdots, p_{n-1})$，规定$p_i = p_{i \bmod n}$
- 找一对i, j，如果$w(p_{i-1}, p_j) + w(p_i, p_{j+1}) < w(p_{i-1}, p_i) + w(p_j, p_{j+1})$则反转$p_i \cdots p_j$一段
- 当没有满足条件的i, j时结束
- 多次重复以上过程取最优

例：旅行商问题

让我们尝试一下爬山算法

- 随机生成初始回路$(p_0, p_1, \cdots, p_{n-1})$，规定$p_i = p_{i \bmod n}$
- 找一对i, j，如果$w(p_{i-1}, p_j) + w(p_i, p_{j+1}) < w(p_{i-1}, p_i) + w(p_j, p_{j+1})$则反转$p_i \cdots p_j$一段
- 当没有满足条件的i, j时结束
- 多次重复以上过程取最优

例：旅行商问题

让我们尝试一下爬山算法

- 随机生成初始回路$(p_0, p_1, \cdots, p_{n-1})$，规定$p_i = p_{i \bmod n}$
- 找一对i, j，如果$w(p_{i-1}, p_j) + w(p_i, p_{j+1}) < w(p_{i-1}, p_i) + w(p_j, p_{j+1})$则反转$p_i \cdots p_j$一段
- 当没有满足条件的i, j时结束
- 多次重复以上过程取最优

例：旅行商问题

让我们尝试一下爬山算法

- 随机生成初始回路$(p_0, p_1, \cdots, p_{n-1})$，规定$p_i = p_{i \bmod n}$
- 找一对i, j，如果$w(p_{i-1}, p_j) + w(p_i, p_{j+1}) < w(p_{i-1}, p_i) + w(p_j, p_{j+1})$则反转$p_i \cdots p_j$一段
- 当没有满足条件的i, j时结束
- 多次重复以上过程取最优

例：旅行商问题

测试情况（TSPLIB数据在我本机上跑一分钟左右的结果）：

数据	目前最优解	爬山算法
$n = 50$	5553	5553
$n = 75$	7054	7059
$n = 100$	7891	7931
$n = 200$	10649	10956
$n = 300$	11865	12302
$n = 400$	14722	15288

例：旅行商问题

遗传算法?

交叉：将两个回路的边放在一起随机打乱，然后贪心加入，最后随机连接每一段

变异：随机交换相邻两个元素

例：旅行商问题

遗传算法?

交叉：将两个回路的边放在一起随机打乱，然后贪心加入，最后随机连接每一段

变异：随机交换相邻两个元素

例：旅行商问题

遗传算法？

交叉：将两个回路的边放在一起随机打乱，然后贪心加入，最后随机连接每一段

变异：随机交换相邻两个元素

例：旅行商问题

遗传算法?

交叉：将两个回路的边放在一起随机打乱，然后贪心加入，最后随机连接每一段

变异：随机交换相邻两个元素

例：旅行商问题

事实证明，遗传算法求解旅行商问题并不优秀......

不仅收敛得慢，结果也不优（也有可能是我的遗传算法写得不好？）

数据	目前最优解	爬山算法	遗传算法
$n = 50$	5553	5553	5952
$n = 75$	7054	7059	8148
$n = 100$	7891	7931	8935
$n = 200$	10649	10956	12879
$n = 300$	11865	12302	14819
$n = 400$	14722	15288	19617

例：旅行商问题

事实证明，遗传算法求解旅行商问题并不优秀......

不仅收敛得慢，结果也不优（也有可能是我的遗传算法写得不好？）

数据	目前最优解	爬山算法	遗传算法
$n = 50$	5553	5553	5952
$n = 75$	7054	7059	8148
$n = 100$	7891	7931	8935
$n = 200$	10649	10956	12879
$n = 300$	11865	12302	14819
$n = 400$	14722	15288	19617

例：旅行商问题

事实证明，遗传算法求解旅行商问题并不优秀......
不仅收敛得慢，结果也不优（也有可能是我的遗传算法写得不好？）

数据	目前最优解	爬山算法	遗传算法
$n = 50$	5553	5553	5952
$n = 75$	7054	7059	8148
$n = 100$	7891	7931	8935
$n = 200$	10649	10956	12879
$n = 300$	11865	12302	14819
$n = 400$	14722	15288	19617

例：旅行商问题

蚁群算法？

当蚂蚁位于i时，对于所有未访问过的结点j，蚂蚁下一步访问j的概率为

$$P_{i,j,S} = \frac{\tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}}{\sum_{k\in S}\tau_{i,k}^{\alpha}\eta_{i,k}^{\beta}}$$

其中$\tau_{i,j}$为(i,j)边上的信息素强度，$\eta_{i,j}$一般为边权的倒数$\frac{1}{w_{i,j}}$，S为未访问的结点集合，α,β为算法参数

例：旅行商问题

蚁群算法?

当蚂蚁位于i时，对于所有未访问过的结点j，蚂蚁下一步访问j的概率为

$$P_{i,j,S} = \frac{\tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}}{\sum_{k\in S}\tau_{i,k}^{\alpha}\eta_{i,k}^{\beta}}$$

其中$\tau_{i,j}$为(i,j)边上的信息素强度，$\eta_{i,j}$一般为边权的倒数$\frac{1}{w_{i,j}}$，S为未访问的结点集合，α,β为算法参数

例：旅行商问题

蚁群算法?

当蚂蚁位于i时，对于所有未访问过的结点j，蚂蚁下一步访问j的概率为

$$P_{i,j,S}=\frac{\tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}}{\sum_{k\in S}\tau_{i,k}^{\alpha}\eta_{i,k}^{\beta}}$$

其中$\tau_{i,j}$为(i,j)边上的信息素强度，$\eta_{i,j}$一般为边权的倒数$\frac{1}{w_{i,j}}$，S为未访问的结点集合，α,β为算法参数

例：旅行商问题

N只蚂蚁访问完之后，修改每只蚂蚁的访问路径的信息素

记$\Delta\tau_{i,j}^k$为第k只蚂蚁在边(i,j)上留下的信息素强度，当蚂蚁不经过边(i,j)时为0，否则调整信息素有多种计算方法：

- $\Delta\tau_{i,j}^k = \frac{Q}{L_k}$，$L_k$为蚂蚁$k$走的路径长度
- $\Delta\tau_{i,j}^k = \frac{Q}{d_{i,j}}$
- $\Delta\tau_{i,j}^k = Q$

循环结束时

$$\tau_{i,j} \leftarrow \rho\tau_{i,j} + \sum_k \Delta\tau_{i,j}^k$$

ρ, Q为常数

例：旅行商问题

N只蚂蚁访问完之后，修改每只蚂蚁的访问路径的信息素

记$\Delta\tau_{i,j}^{k}$为第k只蚂蚁在边(i,j)上留下的信息素强度，当蚂蚁不经过边(i,j)时为0，否则调整信息素有多种计算方法：

- $\Delta\tau_{i,j}^{k}=\frac{Q}{L_k}$，$L_k$为蚂蚁$k$走的路径长度
- $\Delta\tau_{i,j}^{k}=\frac{Q}{d_{i,j}}$
- $\Delta\tau_{i,j}^{k}=Q$

循环结束时

$$\tau_{i,j}\leftarrow\rho\tau_{i,j}+\sum_{k}\Delta\tau_{i,j}^{k}$$

ρ,Q为常数

例：旅行商问题

N只蚂蚁访问完之后，修改每只蚂蚁的访问路径的信息素

记$\Delta\tau_{i,j}^k$为第k只蚂蚁在边(i,j)上留下的信息素强度，当蚂蚁不经过边(i,j)时为0，否则调整信息素有多种计算方法：

- $\Delta\tau_{i,j}^k = \frac{Q}{L_k}$，$L_k$为蚂蚁$k$走的路径长度
- $\Delta\tau_{i,j}^k = \frac{Q}{d_{i,j}}$
- $\Delta\tau_{i,j}^k = Q$

循环结束时

$$\tau_{i,j} \leftarrow \rho\tau_{i,j} + \sum_k \Delta\tau_{i,j}^k$$

ρ, Q为常数

例：旅行商问题

N只蚂蚁访问完之后，修改每只蚂蚁的访问路径的信息素

记$\Delta\tau_{i,j}^k$为第k只蚂蚁在边(i,j)上留下的信息素强度，当蚂蚁不经过边(i,j)时为0，否则调整信息素有多种计算方法：

- $\Delta\tau_{i,j}^k = \frac{Q}{L_k}$，$L_k$为蚂蚁$k$走的路径长度
- $\Delta\tau_{i,j}^k = \frac{Q}{d_{i,j}}$
- $\Delta\tau_{i,j}^k = Q$

循环结束时

$$\tau_{i,j} \leftarrow \rho\tau_{i,j} + \sum_k \Delta\tau_{i,j}^k$$

ρ, Q为常数

例：旅行商问题

N只蚂蚁访问完之后，修改每只蚂蚁的访问路径的信息素

记$\Delta\tau_{i,j}^k$为第k只蚂蚁在边(i,j)上留下的信息素强度，当蚂蚁不经过边(i,j)时为0，否则调整信息素有多种计算方法：

- $\Delta\tau_{i,j}^k = \frac{Q}{L_k}$，$L_k$为蚂蚁$k$走的路径长度
- $\Delta\tau_{i,j}^k = \frac{Q}{d_{i,j}}$
- $\Delta\tau_{i,j}^k = Q$

循环结束时

$$\tau_{i,j} \leftarrow \rho\tau_{i,j} + \sum_k \Delta\tau_{i,j}^k$$

ρ, Q为常数

例：旅行商问题

在之前的爬山算法中，每次交换两条边的方法称为2-OPT
类似的，还有3-OPT、……、k-OPT……单次交换复杂度$O(n^k)$

例：旅行商问题

在之前的爬山算法中，每次交换两条边的方法称为2-OPT
类似的，还有3-OPT、……、k-OPT……单次交换复杂度$O(n^k)$

例：旅行商问题

Lin–Kernighan（LK）算法：一种可变的λ-OPT

- 初始$X = Y = \varnothing$
- 不断把边加入边集X, Y，加入k条边以后考虑加入第$k+1$条边
- 要求当前回路删除X中的边再加入Y中的边总长变小

例：旅行商问题

Lin–Kernighan（LK）算法：一种可变的λ-OPT

- 初始$X = Y = \varnothing$
- 不断把边加入边集X, Y，加入k条边以后考虑加入第$k+1$条边
- 要求当前回路删除X中的边再加入Y中的边总长变小

例：旅行商问题

Lin–Kernighan（LK）算法：一种可变的λ-OPT

- 初始$X = Y = \varnothing$
- 不断把边加入边集X, Y，加入k条边以后考虑加入第$k+1$条边
- 要求当前回路删除X中的边再加入Y中的边总长变小

例：旅行商问题

Lin–Kernighan（LK）算法：一种可变的λ-OPT

- 初始$X = Y = \varnothing$
- 不断把边加入边集X, Y，加入k条边以后考虑加入第$k+1$条边
- 要求当前回路删除X中的边再加入Y中的边总长变小

例：旅行商问题

Lin–Kernighan（LK）算法：一种可变的λ-OPT

- 初始$X = Y = \varnothing$
- 不断把边加入边集X, Y，加入k条边以后考虑加入第$k+1$条边
- 要求当前回路删除X中的边再加入Y中的边总长变小

例：旅行商问题

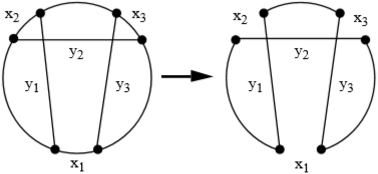


Figure 3.1 A 3-opt move

例：旅行商问题

1. Generate a random initial tour T.
2. Let i = 1. Choose t_1.
3. Choose $x_1 = (t_1,t_2) \in T$.
4. Choose $y_1 = (t_2,t_3) \notin T$ such that $G_1 > 0$.
   If this is not possible, go to Step 12.
5. Let i = i+1.
6. Choose $x_i = (t_{2i-1},t_{2i}) \in T$ such that
   (a) if t_{2i} is joined to t_1, the resulting configuration is a tour, T', and
   (b) $x_i \neq y_s$ for all $s < i$.
   If T' is a better tour than T, let T = T' and go to Step 2.
7. Choose $y_i = (t_{2i},t_{2i+1}) \notin T$ such that
   (a) $G_i > 0$,
   (b) $y_i \neq x_s$ for all $s \leq i$, and
   (c) x_{i+1} exists.
   If such y_i exists, go to Step 5.
8. If there is an untried alternative for y_2, let i = 2 and go to Step 7.
9. If there is an untried alternative for x_2, let i = 2 and go to Step 6.
10. If there is an untried alternative for y_1, let i = 1 and go to Step 4.
11. If there is an untried alternative for x_1, let i = 1 and go to Step 3.
12. If there is an untried alternative for t_1, then go to Step 2.
13. Stop (or go to Step 1).

例：旅行商问题

设$X=\{x_1,x_2,\cdots,x_r\}$，$Y=\{y_1,y_2,\cdots,y_r\}$

优化原则：

- 加入X,Y的边构成回路$(v_1,v_2,\cdots,v_{2r})$，即

$$x_i=(v_{2i-1},v_{2i}),y_i=(v_{2i},v_{2i+1}),v_{2k+1}=v_1$$

- 路径长度的改进量的前缀和为正，即对任意i，

$$\sum_{j=1}^{i}w(x_j)>\sum_{j=1}^{i}w(y_j)$$

- $X\cap Y=\varnothing$
- 边$y_i=(v_{2i},v_{2i+1})$是离v_{2i}最近的5条边之一
- 当新的回路和旧的相同时结束该过程
- ……

例：旅行商问题

设$X = \{x_1, x_2, \cdots, x_r\}$，$Y = \{y_1, y_2, \cdots, y_r\}$
优化原则：

- 加入X, Y的边构成回路$(v_1, v_2, \cdots, v_{2r})$，即

$$x_i = (v_{2i-1}, v_{2i}), y_i = (v_{2i}, v_{2i+1}), v_{2k+1} = v_1$$

- 路径长度的改进量的前缀和为正，即对任意i，

$$\sum_{j=1}^{i} w(x_j) > \sum_{j=1}^{i} w(y_j)$$

- $X \cap Y = \varnothing$
- 边$y_i = (v_{2i}, v_{2i+1})$是离v_{2i}最近的5条边之一
- 当新的回路和旧的相同时结束该过程
- ……

例：旅行商问题

设$X = \{x_1, x_2, \cdots, x_r\}$，$Y = \{y_1, y_2, \cdots, y_r\}$
优化原则：

- 加入X, Y的边构成回路$(v_1, v_2, \cdots, v_{2r})$，即

$$x_i = (v_{2i-1}, v_{2i}), y_i = (v_{2i}, v_{2i+1}), v_{2k+1} = v_1$$

- 路径长度的改进量的前缀和为正，即对任意i，

$$\sum_{j=1}^{i} w(x_j) > \sum_{j=1}^{i} w(y_j)$$

- $X \cap Y = \varnothing$
- 边$y_i = (v_{2i}, v_{2i+1})$是离v_{2i}最近的5条边之一
- 当新的回路和旧的相同时结束该过程
- ……

例：旅行商问题

设$X = \{x_1, x_2, \cdots, x_r\}$，$Y = \{y_1, y_2, \cdots, y_r\}$
优化原则：

- 加入X, Y的边构成回路$(v_1, v_2, \cdots, v_{2r})$，即
$$x_i = (v_{2i-1}, v_{2i}), y_i = (v_{2i}, v_{2i+1}), v_{2k+1} = v_1$$
- 路径长度的改进量的前缀和为正，即对任意i，
$$\sum_{j=1}^{i} w(x_j) > \sum_{j=1}^{i} w(y_j)$$
- $X \cap Y = \varnothing$
- 边$y_i = (v_{2i}, v_{2i+1})$是离v_{2i}最近的5条边之一
- 当新的回路和旧的相同时结束该过程
- ……

例：旅行商问题

设$X = \{x_1, x_2, \cdots, x_r\}$，$Y = \{y_1, y_2, \cdots, y_r\}$
优化原则：

- 加入X, Y的边构成回路$(v_1, v_2, \cdots, v_{2r})$，即

$$x_i = (v_{2i-1}, v_{2i}), y_i = (v_{2i}, v_{2i+1}), v_{2k+1} = v_1$$

- 路径长度的改进量的前缀和为正，即对任意i，

$$\sum_{j=1}^{i} w(x_j) > \sum_{j=1}^{i} w(y_j)$$

- $X \cap Y = \varnothing$
- 边$y_i = (v_{2i}, v_{2i+1})$是离v_{2i}最近的5条边之一
- 当新的回路和旧的相同时结束该过程
- ……

例：旅行商问题

设$X=\{x_1,x_2,\cdots,x_r\}$，$Y=\{y_1,y_2,\cdots,y_r\}$
优化原则：

- 加入X,Y的边构成回路$(v_1,v_2,\cdots,v_{2r})$，即
$$x_i=(v_{2i-1},v_{2i}),y_i=(v_{2i},v_{2i+1}),v_{2k+1}=v_1$$
- 路径长度的改进量的前缀和为正，即对任意i，
$$\sum_{j=1}^{i}w(x_j)>\sum_{j=1}^{i}w(y_j)$$
- $X\cap Y=\varnothing$
- 边$y_i=(v_{2i},v_{2i+1})$是离v_{2i}最近的5条边之一
- 当新的回路和旧的相同时结束该过程
- ......

例：旅行商问题

设$X = \{x_1, x_2, \cdots, x_r\}$，$Y = \{y_1, y_2, \cdots, y_r\}$
优化原则：

- 加入X, Y的边构成回路$(v_1, v_2, \cdots, v_{2r})$，即
$$x_i = (v_{2i-1}, v_{2i}), y_i = (v_{2i}, v_{2i+1}), v_{2k+1} = v_1$$
- 路径长度的改进量的前缀和为正，即对任意i，
$$\sum_{j=1}^{i} w(x_j) > \sum_{j=1}^{i} w(y_j)$$
- $X \cap Y = \varnothing$
- 边$y_i = (v_{2i}, v_{2i+1})$是离v_{2i}最近的5条边之一
- 当新的回路和旧的相同时结束该过程
- ……

例：旅行商问题

LK算法的效果比较优秀，不过并不是最优秀的
Lin-Kernighan-Helsgaun（LKH）算法：LK算法的改进
有兴趣的同学可以了解一下

例：旅行商问题

LK算法的效果比较优秀，不过并不是最优秀的

Lin-Kernighan-Helsgaun（LKH）算法：LK算法的改进

有兴趣的同学可以了解一下

例：旅行商问题

LK算法的效果比较优秀，不过并不是最优秀的

Lin-Kernighan-Helsgaun（LKH）算法：LK算法的改进

有兴趣的同学可以了解一下

例：平面图旅行商问题

旅行商问题的特殊形式：

- n个节点，每个节点有一个坐标(x_i, y_i)
- 节点i,j之间的边权$w_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

平面旅行商问题的输入规模为$O(n)$，因此可以有较大（如$n = 100000$）的数据规模，即使贪心找最近邻的$O(n^2)$近似算法都需要花费较多时间

例：平面图旅行商问题

旅行商问题的特殊形式：

- n个节点，每个节点有一个坐标(x_i, y_i)
- 节点i,j之间的边权$w_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

平面旅行商问题的输入规模为$O(n)$，因此可以有较大（如$n = 100000$）的数据规模，即使贪心找最近邻的$O(n^2)$近似算法都需要花费较多时间

例：平面图旅行商问题

旅行商问题的特殊形式：

- n个节点，每个节点有一个坐标(x_i, y_i)
- 节点i, j之间的边权$w_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

平面旅行商问题的输入规模为$O(n)$，因此可以有较大（如$n = 100000$）的数据规模，即使贪心找最近邻的$O(n^2)$近似算法都需要花费较多时间

例：平面图旅行商问题

旅行商问题的特殊形式：

- n个节点，每个节点有一个坐标(x_i, y_i)
- 节点i, j之间的边权$w_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

平面旅行商问题的输入规模为$O(n)$，因此可以有较大（如$n = 100000$）的数据规模，即使贪心找最近邻的$O(n^2)$近似算法都需要花费较多时间

例：平面图旅行商问题

- 将平面等分成左上、右上、左下、右下四个区域?
- 对每部分递归构造，拼接起来?
- 假设两个相邻区域之间的边界最多跨越c次?

通过增大c的值、多次随机切割（对于$M \times M$的区域，随机$x_0, y_0 \in [0, \frac{M}{2})$，按照$x = x_0$、$x = x_0 + \frac{M}{2}$、$y = y_0$、$y = y_0 + \frac{M}{2}$切割）构造，可以使得到的解趋近于最优解

证明较为复杂

例：平面图旅行商问题

- 将平面等分成左上、右上、左下、右下四个区域?
- 对每部分递归构造，拼接起来?
- 假设两个相邻区域之间的边界最多跨越c次?

通过增大c的值、多次随机切割（对于$M \times M$的区域，随机$x_0, y_0 \in [0, \frac{M}{2})$，按照$x = x_0$、$x = x_0 + \frac{M}{2}$、$y = y_0$、$y = y_0 + \frac{M}{2}$切割）构造，可以使得到的解趋近于最优解

证明较为复杂

例：平面图旅行商问题

- 将平面等分成左上、右上、左下、右下四个区域?
- 对每部分递归构造，拼接起来?
- 假设两个相邻区域之间的边界最多跨越c次?

通过增大c的值、多次随机切割（对于$M \times M$的区域，随机$x_0, y_0 \in [0, \frac{M}{2})$，按照$x = x_0$、$x = x_0 + \frac{M}{2}$、$y = y_0$、$y = y_0 + \frac{M}{2}$切割）构造，可以使得到的解趋近于最优解

证明较为复杂

例：平面图旅行商问题

- 将平面等分成左上、右上、左下、右下四个区域?
- 对每部分递归构造，拼接起来?
- 假设两个相邻区域之间的边界最多跨越c次?

通过增大c的值、多次随机切割（对于$M \times M$的区域，随机$x_0, y_0 \in [0, \frac{M}{2})$，按照$x = x_0$、$x = x_0 + \frac{M}{2}$、$y = y_0$、$y = y_0 + \frac{M}{2}$切割）构造，可以使得到的解趋近于最优解

证明较为复杂

例：平面图旅行商问题

- 将平面等分成左上、右上、左下、右下四个区域?
- 对每部分递归构造，拼接起来?
- 假设两个相邻区域之间的边界最多跨越c次?

通过增大c的值、多次随机切割（对于$M \times M$的区域，随机$x_0, y_0 \in [0, \frac{M}{2})$，按照$x = x_0$、$x = x_0 + \frac{M}{2}$、$y = y_0$、$y = y_0 + \frac{M}{2}$切割）构造，可以使得到的解趋近于最优解

证明较为复杂

博弈问题

OI中经常遇到二人博弈问题，如取石子游戏、皇后移动游戏等

- 零和博弈：博弈双方的收益和损失的总和为0，一方的收益意味着另一方的损失

例如，双方只有一方获胜的游戏、“Alice想最大化这个分数，而Bob想最小化这个分数”的游戏就是零和的

- 完全信息博弈：博弈双方对另一方的状态有完备的信息

例如围棋是完全信息的，而斗地主不是完全信息的

博弈问题

OI中经常遇到二人博弈问题，如取石子游戏、皇后移动游戏等

- 零和博弈：博弈双方的收益和损失的总和为0，一方的收益意味着另一方的损失

例如，双方只有一方获胜的游戏、"Alice想最大化这个分数，而Bob想最小化这个分数"的游戏就是零和的

- 完全信息博弈：博弈双方对另一方的状态有完备的信息

例如围棋是完全信息的，而斗地主不是完全信息的

博弈问题

OI中经常遇到二人博弈问题，如取石子游戏、皇后移动游戏等

- 零和博弈：博弈双方的收益和损失的总和为0，一方的收益意味着另一方的损失

例如，双方只有一方获胜的游戏、“Alice想最大化这个分数，而Bob想最小化这个分数”的游戏就是零和的

- 完全信息博弈：博弈双方对另一方的状态有完备的信息

例如围棋是完全信息的，而斗地主不是完全信息的

博弈问题

OI中经常遇到二人博弈问题，如取石子游戏、皇后移动游戏等

- 零和博弈：博弈双方的收益和损失的总和为0，一方的收益意味着另一方的损失

例如，双方只有一方获胜的游戏、“Alice想最大化这个分数，而Bob想最小化这个分数”的游戏就是零和的

- 完全信息博弈：博弈双方对另一方的状态有完备的信息

例如围棋是完全信息的，而斗地主不是完全信息的

博弈问题

OI中经常遇到二人博弈问题，如取石子游戏、皇后移动游戏等

- 零和博弈：博弈双方的收益和损失的总和为0，一方的收益意味着另一方的损失

例如，双方只有一方获胜的游戏、“Alice想最大化这个分数，而Bob想最小化这个分数”的游戏就是零和的

- 完全信息博弈：博弈双方对另一方的状态有完备的信息

例如围棋是完全信息的，而斗地主不是完全信息的

博弈问题

OI中经常遇到二人博弈问题，如取石子游戏、皇后移动游戏等

- 零和博弈：博弈双方的收益和损失的总和为0，一方的收益意味着另一方的损失

例如，双方只有一方获胜的游戏、“Alice想最大化这个分数，而Bob想最小化这个分数”的游戏就是零和的

- 完全信息博弈：博弈双方对另一方的状态有完备的信息

例如围棋是完全信息的，而斗地主不是完全信息的

博弈问题

考虑有限状态的二人零和博弈，可以抽象为如下模型：

- 有一个有向图$G=(V,E)$和一个棋子v，初始时v位于s处
- 双方轮流操作，将v沿任意一条边$(v,u)\in E$移动至u
- 每个结点有一个分数$f(v)$，不能移动（即$\deg^{+}(v)=0$）时游戏结束，这一方获得$-f(v)$的分数，对方获得$f(v)$的分数
- 双方希望最大化自己的分数

博弈问题

考虑有限状态的二人零和博弈，可以抽象为如下模型：

- 有一个有向图$G=(V,E)$和一个棋子v，初始时v位于s处
- 双方轮流操作，将v沿任意一条边$(v,u)\in E$移动至u
- 每个结点有一个分数$f(v)$，不能移动（即$\deg^{+}(v)=0$）时游戏结束，这一方获得$-f(v)$的分数，对方获得$f(v)$的分数
- 双方希望最大化自己的分数

博弈问题

考虑有限状态的二人零和博弈，可以抽象为如下模型：

- 有一个有向图$G = (V, E)$和一个棋子v，初始时v位于s处
- 双方轮流操作，将v沿任意一条边$(v, u) \in E$移动至u
- 每个结点有一个分数$f(v)$，不能移动（即$\deg^+(v) = 0$）时游戏结束，这一方获得$-f(v)$的分数，对方获得$f(v)$的分数
- 双方希望最大化自己的分数

博弈问题

考虑有限状态的二人零和博弈，可以抽象为如下模型：

- 有一个有向图$G = (V, E)$和一个棋子v，初始时v位于s处
- 双方轮流操作，将v沿任意一条边$(v, u) \in E$移动至u
- 每个结点有一个分数$f(v)$，不能移动（即$\deg^+(v) = 0$）时游戏结束，这一方获得$-f(v)$的分数，对方获得$f(v)$的分数
- 双方希望最大化自己的分数

博弈问题

考虑有限状态的二人零和博弈，可以抽象为如下模型：

- 有一个有向图$G = (V, E)$和一个棋子v，初始时v位于s处
- 双方轮流操作，将v沿任意一条边$(v, u) \in E$移动至u
- 每个结点有一个分数$f(v)$，不能移动（即$\deg^+(v) = 0$）时游戏结束，这一方获得$-f(v)$的分数，对方获得$f(v)$的分数
- 双方希望最大化自己的分数

博弈问题

OI中的不少博弈问题都有经典结论：

- SG定理
- 无向图游戏$\longrightarrow$最大匹配
- ……

博弈问题

OI中的不少博弈问题都有经典结论：

- SG定理
- 无向图游戏$\longrightarrow$最大匹配
- ……

博弈问题

OI中的不少博弈问题都有经典结论：

- SG定理
- 无向图游戏$\longrightarrow$最大匹配
- ……

博弈问题

OI中的不少博弈问题都有经典结论：

- SG定理
- 无向图游戏$\longrightarrow$最大匹配
- ……

例：Codeforces 850C

给定n个数a_i，先手和后首轮流操作，每次操作选择一个质数p和一个正整数k，要求$\exists i, p^k | a_i$，然后对所有$p^k | a_i$的i，令$a_i \leftarrow \frac{a_i}{p^k}$，不能操作者输

判断先手和后手谁有必胜策略，$n \leq 100$，$a_i \leq 10^9$

例：Codeforces 850C

给定n个数a_i，先手和后首轮流操作，每次操作选择一个质数p和一个正整数k，要求$\exists i, p^k | a_i$，然后对所有$p^k | a_i$的i，令$a_i \leftarrow \frac{a_i}{p^k}$，不能操作者输

判断先手和后手谁有必胜策略，$n \le 100$，$a_i \le 10^9$

例：Codeforces 850C

- 将所有a_i质因数分解
- 记p_i为出现的第i个质数，$x_{i,j}$表示是否存在一个数，p_i的次数为j
- 每次操作相当于选择i,j，令所有k，$x_{i,k} \leftarrow \begin{cases} x_{i,k} \vee x_{i,k+j}, & k < j, \\ x_{i,k+j}, & k \geq j \end{cases}$
- 每个x_i独立，分别计算$SG(x_i)$最后异或起来判断是否为0即可，非0为先手必胜，0为后手必胜

例：Codeforces 850C

- 将所有a_i质因数分解
- 记p_i为出现的第i个质数，$x_{i,j}$表示是否存在一个数，p_i的次数为j
- 每次操作相当于选择i,j，令所有k，$x_{i,k} \leftarrow \begin{cases} x_{i,k} \vee x_{i,k+j}, & k < j, \\ x_{i,k+j}, & k \geq j \end{cases}$
- 每个x_i独立，分别计算$SG(x_i)$最后异或起来判断是否为0即可，非0为先手必胜，0为后手必胜

例：Codeforces 850C

- 将所有a_i质因数分解
- 记p_i为出现的第i个质数，$x_{i,j}$表示是否存在一个数，p_i的次数为j
- 每次操作相当于选择i,j，令所有k，$x_{i,k} \leftarrow \begin{cases} x_{i,k} \vee x_{i,k+j}, & k < j, \\ x_{i,k+j}, & k \geq j \end{cases}$
- 每个x_i独立，分别计算$SG(x_i)$最后异或起来判断是否为0即可，非0为先手必胜，0为后手必胜

例：Codeforces 850C

- 将所有a_i质因数分解
- 记p_i为出现的第i个质数，$x_{i,j}$表示是否存在一个数，p_i的次数为j
- 每次操作相当于选择i,j，令所有k，$x_{i,k} \leftarrow \begin{cases} x_{i,k} \vee x_{i,k+j}, & k < j, \\ x_{i,k+j}, & k \geq j \end{cases}$
- 每个x_i独立，分别计算$SG(x_i)$最后异或起来判断是否为0即可，非0为先手必胜，0为后手必胜

例：Codeforces 850C

- $2^{29} < 10^9$，$3^{19} \geq 10^9$，所以对于除2以外的质数对应的x_i，可以直接状压DP求SG值，复杂度2^{19}
- 对于2，记忆化搜索，如果只记忆化位数$L \leq 20$以内的状态，总状态数可估计为

$$f(L) = \begin{cases} \sum_{i=1}^{L} f(\max\{i-1, L-i\}), & L > 20 \\ 1, & L \leq 20 \end{cases}$$

$f(29)$在10^5级别（实际状态数可能更少），时间效率$2^{20} + f(29)$，可以通过

例：Codeforces 850C

- $2^{29} < 10^9$，$3^{19} \geq 10^9$，所以对于除2以外的质数对应的x_i，可以直接状压DP求SG值，复杂度2^{19}
- 对于2，记忆化搜索，如果只记忆化位数$L \leq 20$以内的状态，总状态数可估计为

$$f(L) = \begin{cases} \sum_{i=1}^{L} f(\max\{i-1, L-i\}), & L > 20 \\ 1, & L \leq 20 \end{cases}$$

$f(29)$在10^5级别（实际状态数可能更少），时间效率$2^{20} + f(29)$，可以通过

博弈问题

然而，更多的博弈问题并没有高效的确定性算法

这时候就需要一些技巧了

博弈问题

然而，更多的博弈问题并没有高效的确定性算法
这时候就需要一些技巧了

估价函数

- 状态数过多，无法穷举所有状态
- 采用估价的方法：搜索到一定深度时，对当前状态估计一个分数，先手越有可能获胜分数越大

估价函数

- 状态数过多，无法穷举所有状态
- 采用估价的方法：搜索到一定深度时，对当前状态估计一个分数，先手越有可能获胜分数越大

极大极小搜索

类比博弈论中的必胜态和必败态：

- 必胜态：至少能转移到一个必败态
- 必败态：转移到的所有状态均为必胜态

可以得到对抗搜索的基本模型

极大极小搜索

类比博弈论中的必胜态和必败态：

- 必胜态：至少能转移到一个必败态
- 必败态：转移到的所有状态均为必胜态

可以得到对抗搜索的基本模型

极大极小搜索

类比博弈论中的必胜态和必败态：

- 必胜态：至少能转移到一个必败态
- 必败态：转移到的所有状态均为必胜态

可以得到对抗搜索的基本模型

极大极小搜索

- 暴力的做法是搜索每一步的决策，搜到某一方时，这一方需要采用对自己最有利的策略
- 搜索深度到达上限或无法行动时，返回估价函数值
- 对于其余结点，假设A要最大化分数，B要最小化分数，在搜索树中，一个结点的值为该状态下最后得到的分数
- 轮到A的结点为Max结点，需要选择分数最大的子结点作为决策
- 轮到B的结点为Min结点，需要选择分数最小的子结点作为决策

极大极小搜索

- 暴力的做法是搜索每一步的决策，搜到某一方时，这一方需要采用对自己最有利的策略
- 搜索深度到达上限或无法行动时，返回估价函数值
- 对于其余结点，假设A要最大化分数，B要最小化分数，在搜索树中，一个结点的值为该状态下最后得到的分数
- 轮到A的结点为Max结点，需要选择分数最大的子结点作为决策
- 轮到B的结点为Min结点，需要选择分数最小的子结点作为决策

极大极小搜索

- 暴力的做法是搜索每一步的决策，搜到某一方时，这一方需要采用对自己最有利的策略
- 搜索深度到达上限或无法行动时，返回估价函数值
- 对于其余结点，假设A要最大化分数，B要最小化分数，在搜索树中，一个结点的值为该状态下最后得到的分数
- 轮到A的结点为Max结点，需要选择分数最大的子结点作为决策
- 轮到B的结点为Min结点，需要选择分数最小的子结点作为决策

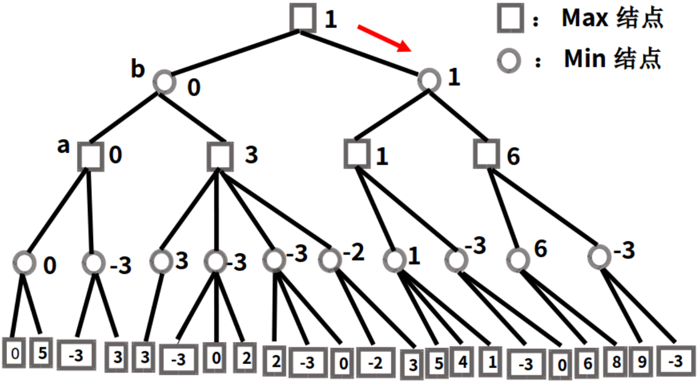
极大极小搜索

- 暴力的做法是搜索每一步的决策，搜到某一方时，这一方需要采用对自己最有利的策略
- 搜索深度到达上限或无法行动时，返回估价函数值
- 对于其余结点，假设A要最大化分数，B要最小化分数，在搜索树中，一个结点的值为该状态下最后得到的分数
- 轮到A的结点为Max结点，需要选择分数最大的子结点作为决策
- 轮到B的结点为Min结点，需要选择分数最小的子结点作为决策

极大极小搜索

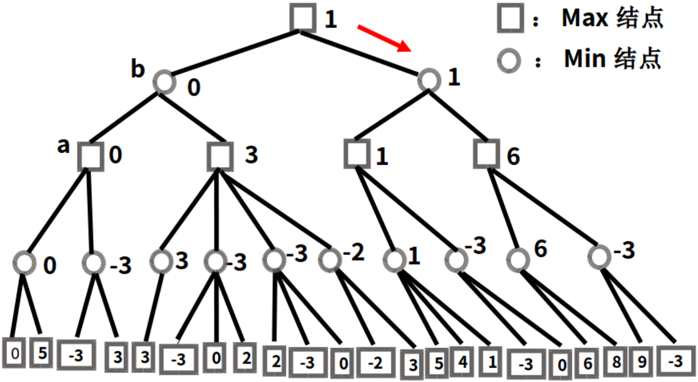
- 暴力的做法是搜索每一步的决策，搜到某一方时，这一方需要采用对自己最有利的策略
- 搜索深度到达上限或无法行动时，返回估价函数值
- 对于其余结点，假设A要最大化分数，B要最小化分数，在搜索树中，一个结点的值为该状态下最后得到的分数
- 轮到A的结点为Max结点，需要选择分数最大的子结点作为决策
- 轮到B的结点为Min结点，需要选择分数最小的子结点作为决策

极大极小搜索



极大极小搜索还有一种实现方式是“负值最大”，一个结点的分数是子结点取相反数后的最大值，可以减少代码量。

极大极小搜索



极大极小搜索还有一种实现方式是“负值最大”，一个结点的分数是子结点取相反数后的最大值，可以减少代码量

Alpha-Beta剪枝

极大极小搜索的最优性剪枝

- 搜索时记录目前祖先结点的Max结点最大值α和Min结点的最小值β
- 当后辈结点的$\alpha \geq$祖先结点的β时剪枝
- 当后辈结点的$\beta \leq$祖先结点的α时剪枝

Alpha-Beta剪枝

极大极小搜索的最优性剪枝

- 搜索时记录目前祖先结点的Max结点最大值α和Min结点的最小值β
- 当后辈结点的$\alpha \geq$祖先结点的β时剪枝
- 当后辈结点的$\beta \leq$祖先结点的α时剪枝

Alpha-Beta剪枝

极大极小搜索的最优性剪枝

- 搜索时记录目前祖先结点的Max结点最大值α和Min结点的最小值β
- 当后辈结点的$\alpha \geq$祖先结点的β时剪枝
- 当后辈结点的$\beta \leq$祖先结点的α时剪枝

Alpha-Beta剪枝

极大极小搜索的最优性剪枝

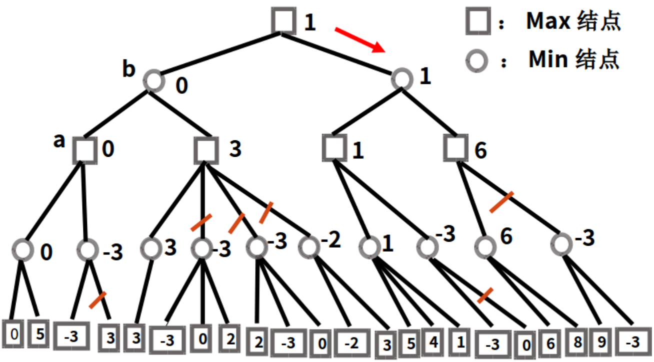
- 搜索时记录目前祖先结点的Max结点最大值α和Min结点的最小值β
- 当后辈结点的$\alpha \geq$祖先结点的β时剪枝
- 当后辈结点的$\beta \leq$祖先结点的α时剪枝

Alpha-Beta剪枝

极大极小搜索的最优性剪枝

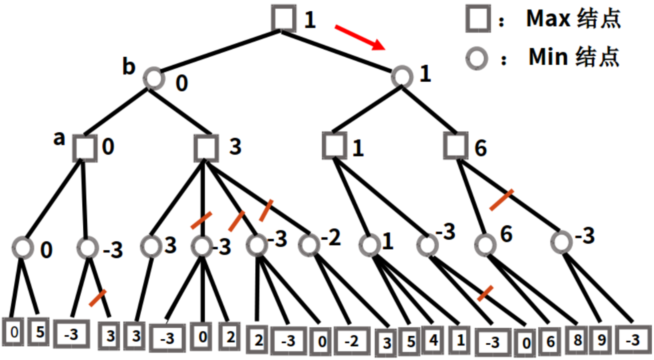
- 搜索时记录目前祖先结点的Max结点最大值α和Min结点的最小值β
- 当后辈结点的$\alpha \geq$祖先结点的β时剪枝
- 当后辈结点的$\beta \leq$祖先结点的α时剪枝

Alpha-Beta剪枝



如果用“负值最大”实现，递归子结点时$(\alpha,\beta)\rightarrow(-\beta,-\alpha)$

Alpha-Beta剪枝



如果用“负值最大”实现，递归子结点时$(\alpha,\beta)\rightarrow(-\beta,-\alpha)$

蒙特卡罗树搜索

然而，对于一些复杂的游戏比如围棋，每一步的走法太多，且评估局面困难

不妨尝试蒙特卡罗方法：

- 考虑以双方随机操作的情况下先手的胜率作为估价
- 进行大量次数的随机模拟来计算估价

这种方法存在怎样的缺陷?

蒙特卡罗树搜索

然而，对于一些复杂的游戏比如围棋，每一步的走法太多，且评估局面困难

不妨尝试蒙特卡罗方法：

- 考虑以双方随机操作的情况下先手的胜率作为估价
- 进行大量次数的随机模拟来计算估价

这种方法存在怎样的缺陷?

蒙特卡罗树搜索

然而，对于一些复杂的游戏比如围棋，每一步的走法太多，且评估局面困难

不妨尝试蒙特卡罗方法：

- 考虑以双方随机操作的情况下先手的胜率作为估价
- 进行大量次数的随机模拟来计算估价

这种方法存在怎样的缺陷?

蒙特卡罗树搜索

然而，对于一些复杂的游戏比如围棋，每一步的走法太多，且评估局面困难

不妨尝试蒙特卡罗方法：

- 考虑以双方随机操作的情况下先手的胜率作为估价
- 进行大量次数的随机模拟来计算估价

这种方法存在怎样的缺陷?

蒙特卡罗树搜索

然而，对于一些复杂的游戏比如围棋，每一步的走法太多，且评估局面困难

不妨尝试蒙特卡罗方法：

- 考虑以双方随机操作的情况下先手的胜率作为估价
- 进行大量次数的随机模拟来计算估价

这种方法存在怎样的缺陷?

蒙特卡罗树搜索

- 假设轮到A行动，A有a_1, a_2两种走法
- 如果A走a_1，B有9种方法应对会很快输，1种方法应对会很快赢
- 如果A走a_2，B没有很快能赢的走法，但如果此时双方随机操作，双方各有50%概率赢
- 那么蒙特卡罗方法A会优先走a_1
- 然而这样是必败的，走a_2更优

蒙特卡罗树搜索

- 假设轮到A行动，A有a_1, a_2两种走法
- 如果A走a_1，B有9种方法应对会很快输，1种方法应对会很快赢
- 如果A走a_2，B没有很快能赢的走法，但如果此时双方随机操作，双方各有50%概率赢
- 那么蒙特卡罗方法A会优先走a_1
- 然而这样是必败的，走a_2更优

蒙特卡罗树搜索

- 假设轮到A行动，A有a_1, a_2两种走法
- 如果A走a_1，B有9种方法应对会很快输，1种方法应对会很快赢
- 如果A走a_2，B没有很快能赢的走法，但如果此时双方随机操作，双方各有50%概率赢
- 那么蒙特卡罗方法A会优先走a_1
- 然而这样是必败的，走a_2更优

蒙特卡罗树搜索

- 假设轮到A行动，A有a_1, a_2两种走法
- 如果A走a_1，B有9种方法应对会很快输，1种方法应对会很快赢
- 如果A走a_2，B没有很快能赢的走法，但如果此时双方随机操作，双方各有50%概率赢
- 那么蒙特卡罗方法A会优先走a_1
- 然而这样是必败的，走a_2更优

蒙特卡罗树搜索

- 假设轮到A行动，A有a_1, a_2两种走法
- 如果A走a_1，B有9种方法应对会很快输，1种方法应对会很快赢
- 如果A走a_2，B没有很快能赢的走法，但如果此时双方随机操作，双方各有50%概率赢
- 那么蒙特卡罗方法A会优先走a_1
- 然而这样是必败的，走a_2更优

蒙特卡罗树搜索

- 假设轮到A行动，A有a_1, a_2两种走法
- 如果A走a_1，B有9种方法应对会很快输，1种方法应对会很快赢
- 如果A走a_2，B没有很快能赢的走法，但如果此时双方随机操作，双方各有50%概率赢
- 那么蒙特卡罗方法A会优先走a_1
- 然而这样是必败的，走a_2更优

蒙特卡罗树搜索

蒙特卡罗树搜索（MCTS）相比蒙特卡罗方法（简单模拟），将随机抽样的状态用状态树来表示，然后用极大极小的方式扩展状态树

同样可以在抽样过程中随时得到各决策的估价

搜索时还可以利用之前的结果，以提高效率

AlphaGo主要采用的算法就是蒙特卡罗树搜索

蒙特卡罗树搜索

蒙特卡罗树搜索（MCTS）相比蒙特卡罗方法（简单模拟），将随机抽样的状态用状态树来表示，然后用极大极小的方式扩展状态树

同样可以在抽样过程中随时得到各决策的估价

搜索时还可以利用之前的结果，以提高效率

AlphaGo主要采用的算法就是蒙特卡罗树搜索

蒙特卡罗树搜索

蒙特卡罗树搜索（MCTS）相比蒙特卡罗方法（简单模拟），将随机抽样的状态用状态树来表示，然后用极大极小的方式扩展状态树

同样可以在抽样过程中随时得到各决策的估价

搜索时还可以利用之前的结果，以提高效率

AlphaGo主要采用的算法就是蒙特卡罗树搜索

蒙特卡罗树搜索

蒙特卡罗树搜索（MCTS）相比蒙特卡罗方法（简单模拟），将随机抽样的状态用状态树来表示，然后用极大极小的方式扩展状态树

同样可以在抽样过程中随时得到各决策的估价

搜索时还可以利用之前的结果，以提高效率

AlphaGo主要采用的算法就是蒙特卡罗树搜索

蒙特卡罗树搜索

- 选择：从根节点出发自上而下地选择一个落子点
- 扩展：向选定的点添加一个或多个子节点
- 模拟：对扩展出的节点用蒙特卡罗方法进行模拟
- 回溯：根据模拟结果依次向上更新祖先节点估计值

蒙特卡罗树搜索

- 选择：从根节点出发自上而下地选择一个落子点
- 扩展：向选定的点添加一个或多个子节点
- 模拟：对扩展出的节点用蒙特卡罗方法进行模拟
- 回溯：根据模拟结果依次向上更新祖先节点估计值

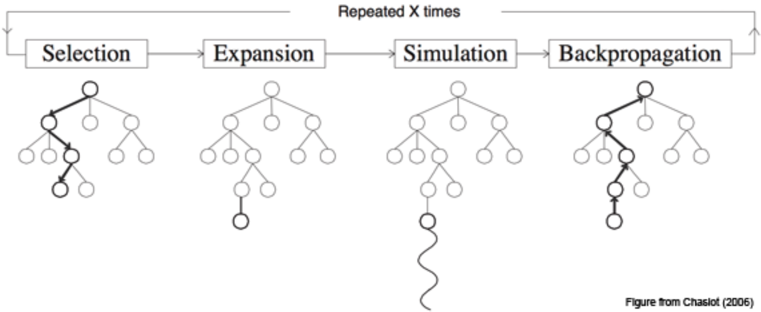
蒙特卡罗树搜索

- 选择：从根节点出发自上而下地选择一个落子点
- 扩展：向选定的点添加一个或多个子节点
- 模拟：对扩展出的节点用蒙特卡罗方法进行模拟
- 回溯：根据模拟结果依次向上更新祖先节点估计值

蒙特卡罗树搜索

- 选择：从根节点出发自上而下地选择一个落子点
- 扩展：向选定的点添加一个或多个子节点
- 模拟：对扩展出的节点用蒙特卡罗方法进行模拟
- 回溯：根据模拟结果依次向上更新祖先节点估计值

蒙特卡罗树搜索


Repeated X times
Selection
Expansion
Simulation
Backpropagation
Figure from Chaslot (2006)

蒙特卡罗树搜索

节点的选择：

优先选择获胜希望较大的分支搜索

简化模型：选择每个落子点的胜率遵循一定概率且互不相关，如何选择一个策略使得胜率最大

信心上限算法（UCB）：

- 对每个落子点访问一次并记录胜利情况
- 然后不断计算所有落子点的信心上界I_j并访问最大的一个，直到达到访问次数上限

蒙特卡罗树搜索

节点的选择：
优先选择获胜希望较大的分支搜索

简化模型：选择每个落子点的胜率遵循一定概率且互不相关，如何选择一个策略使得胜率最大
信心上限算法（UCB）：

- 对每个落子点访问一次并记录胜利情况
- 然后不断计算所有落子点的信心上界I_j并访问最大的一个，直到达到访问次数上限

蒙特卡罗树搜索

节点的选择：

优先选择获胜希望较大的分支搜索

简化模型：选择每个落子点的胜率遵循一定概率且互不相关，如何选择一个策略使得胜率最大

信心上限算法（UCB）：

- 对每个落子点访问一次并记录胜利情况
- 然后不断计算所有落子点的信心上界I_j并访问最大的一个，直到达到访问次数上限

蒙特卡罗树搜索

节点的选择：
优先选择获胜希望较大的分支搜索
简化模型：选择每个落子点的胜率遵循一定概率且互不相关，如何选择一个策略使得胜率最大
信心上限算法（UCB）：

- 对每个落子点访问一次并记录胜利情况
- 然后不断计算所有落子点的信心上界I_i并访问最大的一个，直到达到访问次数上限

蒙特卡罗树搜索

节点的选择：
优先选择获胜希望较大的分支搜索
简化模型：选择每个落子点的胜率遵循一定概率且互不相关，如何选择一个策略使得胜率最大
信心上限算法（UCB）：

- 对每个落子点访问一次并记录胜利情况
- 然后不断计算所有落子点的信心上界I_j并访问最大的一个，直到达到访问次数上限

蒙特卡罗树搜索

节点的选择：
优先选择获胜希望较大的分支搜索
简化模型：选择每个落子点的胜率遵循一定概率且互不相关，如何选择一个策略使得胜率最大
信心上限算法（UCB）：

- 对每个落子点访问一次并记录胜利情况
- 然后不断计算所有落子点的信心上界I_j并访问最大的一个，直到达到访问次数上限

蒙特卡罗树搜索

信心上界I_j的确定：

$$I_j = \overline{X_j} + \sqrt{\frac{2\ln n}{T_j(n)}}$$

- $\overline{X_j}$：决策j当前收益的均值
- n：到当前为止访问的总次数
- $T_j(n)$：决策j到当前位置访问的次数

实际计算时，设定一个参数c，定义

$$I_j = \overline{X_j} + c\sqrt{\frac{2\ln n}{T_j(n)}}$$

UCB公式对开发和探索进行了平衡

蒙特卡罗树搜索

信心上界I_j的确定：

$$I_j = \overline{X_j} + \sqrt{\frac{2 \ln n}{T_j(n)}}$$

- $\overline{X_j}$：决策j当前收益的均值
- n：到当前为止访问的总次数
- $T_j(n)$：决策j到当前位置访问的次数

实际计算时，设定一个参数c，定义

$$I_j = \overline{X_j} + c\sqrt{\frac{2 \ln n}{T_j(n)}}$$

UCB公式对开发和探索进行了平衡

蒙特卡罗树搜索

信心上界I_j的确定：

$$I_j = \overline{X_j} + \sqrt{\frac{2\ln n}{T_j(n)}}$$

- $\overline{X_j}$：决策j当前收益的均值
- n：到当前为止访问的总次数
- $T_j(n)$：决策j到当前位置访问的次数

实际计算时，设定一个参数c，定义

$$I_j = \overline{X_j} + c\sqrt{\frac{2\ln n}{T_j(n)}}$$

UCB公式对开发和探索进行了平衡

蒙特卡罗树搜索

信心上界I_j的确定：

$$I_j = \overline{X_j} + \sqrt{\frac{2\ln n}{T_j(n)}}$$

- $\overline{X_j}$：决策j当前收益的均值
- n：到当前为止访问的总次数
- $T_j(n)$：决策j到当前位置访问的次数

实际计算时，设定一个参数c，定义

$$I_j = \overline{X_j} + c\sqrt{\frac{2\ln n}{T_j(n)}}$$

UCB公式对开发和探索进行了平衡

蒙特卡罗树搜索

信心上界I_j的确定：

$$I_j = \overline{X_j} + \sqrt{\frac{2\ln n}{T_j(n)}}$$

- $\overline{X_j}$：决策j当前收益的均值
- n：到当前为止访问的总次数
- $T_j(n)$：决策j到当前位置访问的次数

实际计算时，设定一个参数c，定义

$$I_j = \overline{X_j} + c\sqrt{\frac{2\ln n}{T_j(n)}}$$

UCB公式对开发和探索进行了平衡

蒙特卡罗树搜索

信心上界I_j的确定：

$$I_j = \overline{X_j} + \sqrt{\frac{2\ln n}{T_j(n)}}$$

- $\overline{X_j}$：决策j当前收益的均值
- n：到当前为止访问的总次数
- $T_j(n)$：决策j到当前位置访问的次数

实际计算时，设定一个参数c，定义

$$I_j = \overline{X_j} + c\sqrt{\frac{2\ln n}{T_j(n)}}$$

UCB公式对开发和探索进行了平衡

蒙特卡罗树搜索

信心上限树算法（UCT）：蒙特卡罗树算法（MCTS）+信息上限算法（UCB）

对每个结点v记录随机模拟的次数$N(v)$和总收益$Q(v)$，大量重复该过程：

- 从根结点出发，每次选择一个信心上限最大（如果$Q(j)=0$则认为$I_j=+\infty$）的子结点往下递归，直到走到未扩展的结点l
- 扩展当前结点l，然后从l开始随机模拟，得到收益Δ
- 回溯时，l的祖先（包括l）的$N(v)\leftarrow N(v)+1$，和l同类型（同为Max或同为Min）的祖先（包括l），$Q(v)\leftarrow Q(v)+\Delta$，非同类祖先$Q(v)\leftarrow Q(v)-\Delta$

通常胜为$\Delta=1$，负为$\Delta=-1$

蒙特卡罗树搜索

信心上限树算法（UCT）：蒙特卡罗树算法（MCTS）+信息上限算法（UCB）

对每个结点v记录随机模拟的次数$N(v)$和总收益$Q(v)$，大量重复该过程：

- 从根结点出发，每次选择一个信心上限最大（如果$Q(j)=0$则认为$I_j=+\infty$）的子结点往下递归，直到走到未扩展的结点l
- 扩展当前结点l，然后从l开始随机模拟，得到收益Δ
- 回溯时，l的祖先（包括l）的$N(v)\leftarrow N(v)+1$，和l同类型（同为Max或同为Min）的祖先（包括l），$Q(v)\leftarrow Q(v)+\Delta$，非同类祖先$Q(v)\leftarrow Q(v)-\Delta$

通常胜为$\Delta=1$，负为$\Delta=-1$

蒙特卡罗树搜索

信心上限树算法（UCT）：蒙特卡罗树算法（MCTS）+信息上限算法（UCB）

对每个结点v记录随机模拟的次数$N(v)$和总收益$Q(v)$，大量重复该过程：

- 从根结点出发，每次选择一个信心上限最大（如果$Q(j)=0$则认为$I_j=+\infty$）的子结点往下递归，直到走到未扩展的结点l
- 扩展当前结点l，然后从l开始随机模拟，得到收益Δ
- 回溯时，l的祖先（包括l）的$N(v)\leftarrow N(v)+1$，和l同类型（同为Max或同为Min）的祖先（包括l），$Q(v)\leftarrow Q(v)+\Delta$，非同类祖先$Q(v)\leftarrow Q(v)-\Delta$

通常胜为$\Delta=1$，负为$\Delta=-1$

蒙特卡罗树搜索

信心上限树算法（UCT）：蒙特卡罗树算法（MCTS）+信息上限算法（UCB）

对每个结点v记录随机模拟的次数$N(v)$和总收益$Q(v)$，大量重复该过程：

- 从根结点出发，每次选择一个信心上限最大（如果$Q(j)=0$则认为$I_j=+\infty$）的子结点往下递归，直到走到未扩展的结点l
- 扩展当前结点l，然后从l开始随机模拟，得到收益Δ
- 回溯时，l的祖先（包括l）的$N(v)\leftarrow N(v)+1$，和l同类型（同为Max或同为Min）的祖先（包括l），$Q(v)\leftarrow Q(v)+\Delta$，非同类祖先$Q(v)\leftarrow Q(v)-\Delta$

通常胜为$\Delta=1$，负为$\Delta=-1$

蒙特卡罗树搜索

信心上限树算法（UCT）：蒙特卡罗树算法（MCTS）+信息上限算法（UCB）

对每个结点v记录随机模拟的次数$N(v)$和总收益$Q(v)$，大量重复该过程：

- 从根结点出发，每次选择一个信心上限最大（如果$Q(j)=0$则认为$I_j=+\infty$）的子结点往下递归，直到走到未扩展的结点l
- 扩展当前结点l，然后从l开始随机模拟，得到收益Δ
- 回溯时，l的祖先（包括l）的$N(v)\leftarrow N(v)+1$，和l同类型（同为Max或同为Min）的祖先（包括l），$Q(v)\leftarrow Q(v)+\Delta$，非同类祖先$Q(v)\leftarrow Q(v)-\Delta$

通常胜为$\Delta=1$，负为$\Delta=-1$

蒙特卡罗树搜索

信心上限树算法（UCT）：蒙特卡罗树算法（MCTS）+信息上限算法（UCB）

对每个结点v记录随机模拟的次数$N(v)$和总收益$Q(v)$，大量重复该过程：

- 从根结点出发，每次选择一个信心上限最大（如果$Q(j)=0$则认为$I_j=+\infty$）的子结点往下递归，直到走到未扩展的结点l
- 扩展当前结点l，然后从l开始随机模拟，得到收益Δ
- 回溯时，l的祖先（包括l）的$N(v)\leftarrow N(v)+1$，和l同类型（同为Max或同为Min）的祖先（包括l），$Q(v)\leftarrow Q(v)+\Delta$，非同类祖先$Q(v)\leftarrow Q(v)-\Delta$

通常胜为$\Delta=1$，负为$\Delta=-1$

蒙特卡罗树搜索

信心上限树算法（UCT）：蒙特卡罗树算法（MCTS）+信息上限算法（UCB）

对每个结点v记录随机模拟的次数$N(v)$和总收益$Q(v)$，大量重复该过程：

- 从根结点出发，每次选择一个信心上限最大（如果$Q(j)=0$则认为$I_j=+\infty$）的子结点往下递归，直到走到未扩展的结点l
- 扩展当前结点l，然后从l开始随机模拟，得到收益Δ
- 回溯时，l的祖先（包括l）的$N(v)\leftarrow N(v)+1$，和l同类型（同为Max或同为Min）的祖先（包括l），$Q(v)\leftarrow Q(v)+\Delta$，非同类祖先$Q(v)\leftarrow Q(v)-\Delta$

通常胜为$\Delta=1$，负为$\Delta=-1$

蒙特卡罗树搜索

- 随着模拟次数的增加，蒙特卡罗树会趋向于往最优解方向生长
- 蒙特卡罗方法有偏差，但蒙特卡罗树搜索没有偏差，只有方差

蒙特卡罗树搜索

- 随着模拟次数的增加，蒙特卡罗树会趋向于往最优解方向生长
- 蒙特卡罗方法有偏差，但蒙特卡罗树搜索没有偏差，只有方差

非多项式复杂度OI题选讲

如果这节课还有时间，放几道原创OI题让大家思考

构造数列

给定长度$n \leq 100$的数字串s，$0 \leq s_i \leq 9$，构造一个长度l尽可能大的正整数数列a_i（$\leq n$）使得

- $a_1 = 1, a_l = n$
- 所有a_i互不相同
- $\forall i, |a_{i+1} - a_i| = s_{a_i}$

构造数列

考虑i向$i+s_i$连一条边，求最长路

注意到s_i很小，状压DP

确定每个点是否属于路径，记$f(i,S)$表示确定了前i个点，其中点$i-8\cdots i$在前面的连接情况为S，剩余的方案数

- 思考：搜索可以过随机数据，如何卡掉搜索?

构造数列

考虑i向$i+s_i$连一条边，求最长路
注意到s_i很小，状压DP
确定每个点是否属于路径，记$f(i,S)$表示确定了前i个点，其中点$i-8\cdots i$在前面的连接情况为S，剩余的方案数

- 思考：搜索可以过随机数据，如何卡掉搜索？

构造数列

考虑i向$i+s_i$连一条边，求最长路

注意到s_i很小，状压DP

确定每个点是否属于路径，记$f(i,S)$表示确定了前i个点，其中点$i-8\cdots i$在前面的连接情况为S，剩余的方案数

- 思考：搜索可以过随机数据，如何卡掉搜索？

构造数列

考虑i向$i+s_i$连一条边，求最长路
注意到s_i很小，状压DP
确定每个点是否属于路径，记$f(i,S)$表示确定了前i个点，其中点$i-8\cdots i$在前面的连接情况为S，剩余的方案数

- 思考：搜索可以过随机数据，如何卡掉搜索？

设计图案

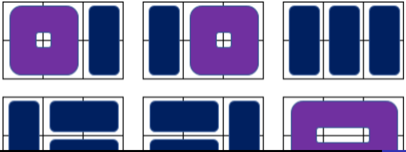
给一个$n \times m$的带障碍网格图，用若干个条形或环形恰好覆盖每个非障碍格子一次

- 条形：覆盖两个有公共边的格子；称这两个格子相连
- 环形：覆盖$c(c \geq 4)$个不同的格子$A_1, A_2, \cdots, A_c$，其中A_i和A_{i+1}（包括A_c和A_1）有公共边；称A_i和A_{i+1}（包括A_c和A_1）相连

一种方案的美观度为2^t，t为环形个数

两种方案不同当且仅当一对格子在一种方案中相连而另一种方案不相连

求所有方案的美观度之和模p的值，$n \times m \leq 300$，$p < 2^{30}$



结论：答案为只用条形覆盖的方案数的平方

想一想，为什么

状压DP求网格图的完美匹配数，复杂度$O(2^{\min\{n,m\}}nm)$

结论：答案为只用条形覆盖的方案数的平方

想一想，为什么

状压DP求网格图的完美匹配数，复杂度$O(2^{\min\{n,m\}}nm)$

结论：答案为只用条形覆盖的方案数的平方
想一想，为什么
状压DP求网格图的完美匹配数，复杂度$O(2^{\min\{n,m\}}nm)$

收集金币

提交答案题

马路上有n堆金币，第i堆位于x_i处，该堆金币有c_i个

有m个机器人，第i个机器人可以收集$[l_i, r_i]$区间内的金币，消耗代价e_i

用不超过h的总代价获得尽量多的金币

$n \le 10^5$，$m \le 10^4$

收集金币

确定性做法:离散化+DP，区间按左端点排序，$f(i,j)$表示选择第i个区间之前的区间，且必须选择第i个区间，最多消耗j的代价，可以获得的金币数

转移：枚举上一个选择的区间（不能包含于第i个区间）

由于是提交答案题，用类似背包的近似算法（把代价或金币数除掉一个常数然后DP）即可

收集金币

确定性做法:离散化+DP，区间按左端点排序，$f(i,j)$表示选择第i个区间之前的区间，且必须选择第i个区间，最多消耗j的代价，可以获得的金币数

转移：枚举上一个选择的区间（不能包含于第i个区间）

由于是提交答案题，用类似背包的近似算法（把代价或金币数除掉一个常数然后DP）即可

收集金币

确定性做法:离散化+DP，区间按左端点排序，$f(i,j)$表示选择第i个区间之前的区间，且必须选择第i个区间，最多消耗j的代价，可以获得的金币数

转移：枚举上一个选择的区间（不能包含于第i个区间）

由于是提交答案题，用类似背包的近似算法（把代价或金币数除掉一个常数然后DP）即可

Steve的编号

给定n阶无向图$G=(V,E)$，给每个点v分配一个$[1,n]$内且互不相同的编号$d(v)$，使得$\max_{(u,v)\in E}\{d(u)-d(v)\}$最小

数据生成方式：随机一个D值，然后随机加入一些满足$|u-v|\le D$的边(u,v)，最后打乱结点编号

开放题，欢迎大家尝试各种做法

Steve的编号

给定n阶无向图$G=(V,E)$，给每个点v分配一个$[1,n]$内且互不相同的编号$d(v)$，使得$\max_{(u,v)\in E}\{d(u)-d(v)\}$最小

数据生成方式：随机一个D值，然后随机加入一些满足$|u-v|\le D$的边(u,v)，最后打乱结点编号

开放题，欢迎大家尝试各种做法