# DP

Future never has to do with past time, but present does.

未来与过去无关，现在决定未来。

问题需要满足无后效性才能用动态规划去解。

## 线性DP

### 最大连续区间和

显然，若序列非负，那越长越好。

必须考虑有正有负的情况。

求最小和同理

以上です。

↑~~不过只适用于写报告，写信千万别用，太冷漠~~

### 最长上升子序列（LIS）

设f[i]表示第位置1到i的以i为结尾的最长上升子序列的长度，那么状态转移方程为：f[i] = max(f[j]) + 1 (j<=i, a[j]<=a[i])，很好理解。

算法竞赛入门经典（第二版）》中未给出优化到的版本，ChatGPT给出了如下代码：

#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  
int LIS(vector<int>& nums) {  
 vector<int> tails;  
 for (int num : nums) {  
 auto it = lower\_bound(tails.begin(), tails.end(), num);  
 //如果不要求严格递增，即求最长不下降子序列，替换为upper\_bound()即可  
 if (it == tails.end()) tails.push\_back(num);  
 else \*it = num;  
 }  
 return tails.size();  
}  
  
int main() {  
 vector<int> nums{10, 9, 2, 5, 3, 7, 101, 18};  
 cout << LIS(nums) << endl; // output: 4  
 return 0;  
}

具体实现时，我们可以使用一个数组tails，其中tails[i]表示长度为i+1的所有LIS中最后一个元素的最小值。遍历原序列，对于每个元素x，使用二分查找在tails数组中找到第一个大于等于x的位置k，更新tails[k]为x，长度为k+1的LIS的末尾元素为x。遍历完整个序列后，tails数组中的最大下标即为LIS的长度。

不难理解，长度越长的最长上升子序列的结尾越大，那么我们可以通过二分查找的方式把第i个元素插入到最长的LIS后面，然后用它的值更该新长度的LIS的结尾元素的最小值。

需要注意的是，求上升子序列用lower\_bound()，而求不下降子序列用upper\_bound()，你可以想想为什么。

### 最长公共子序列

现有两个数列AB，欲求其最长公共子序列。可以在B中找到每个A中元素a[i]第一次在B中出现的位置，若为A中重复的第n个元素就找第n次出现的位置。按照出现位置的先后重新定义数列A的大小关系，求最长不下降子序列即可。

### 01背包

有种物品，每种只有一个。第中物品的体积为，重量为，选一些物品装入容量为的背包，使得背包内物品在总体积不溢出的情况下最大。

设表示“把前个物品装入容量为的背包中的最大总重量，状态转移方程也不难得出：，先枚举i，后枚举j即可。

* 滚动数组

显然，更新只用到了上一行的数据，那么数组第一维长度可以是2。更进一步，第一维可以直接去掉，**但是j要逆序枚举**

for (int i = 1; i <= n; ++i)  
 for (int j = v; j >= 0; --j)  
 f[j] = max(f[j], f[j - c[i]] + w[i]);

## 树形DP

### 树的最大独立集

对于一颗个节点的无根树，选尽量多的节点，使其互不相邻。

只要任选一根r，无根树就变成了有根树。定义表示以为根的子树的最大独立集的大小，那么节点i只两种决策，选或不选。若选，那其儿子都不能选；若不选，那其儿子都可以选。

由此不难得出状态转移方程：。其中表示的孙子，而表示的儿子。

上述方案需要枚举某个节点的儿子和孙子（**填表法**），看起来不是那么优雅。如果计算出一个的值后，用其更新i的父亲和祖父节点的累加值（即$$的两项）（**刷表法**），看起来就优雅许多了。

## 数位DP

数位DP一般解决如下问题：

1. 要求统计满足一定条件的数量（如：计数）；
2. 这些条件经过转化后可以使用「数位」的思想去理解和判断；
3. 输入会提供一个数字区间（有时也只提供上界）来作为统计的限制；
4. 上界很大，暴力枚会超时

一个常用的计数问题的技巧：

那么有了通用答案数组，接下来就是统计答案。统计答案可以选择记忆化搜索，也可以选择循环迭代递推。为了不重不漏地统计所有不超过上限的答案，要从高到低枚举每一位，再考虑每一位都可以填哪些数字，最后利用通用答案数组统计答案。