3121期末复习讲义2

Dynamic Programming

基本思想与分治法类似，也是将待求解的问题分解为若干个子问题（阶段），按顺序求解子阶段，前一子问题的解，为后一子问题的求解提供了有用的信息。在求解任一子问题时，列出各种可能的局部解，通过决策保留那些有可能达到最优的局部解，丢弃其他局部解。依次解决各子问题，最后一个子问题就是初始问题的解。

由于动态规划解决的问题多数有重叠子问题这个特点，为减少重复计算，对每一个子问题只解一次，将其不同阶段的不同状态保存在一个数组中。

1. 找到子问题
2. 写出base case
3. 写出状态转移方程（Recursion）

线性DP （比较灵活）

01背包

一个背包容量C，有n个物品，每个物品i有重量wi，价值pi。每个物品只有一件。如何让我们的背包里面物品价值最高

wi 1 5 10

pi 5 4 12

i（物品序号） j（背包容量）	1	2	3
1	5	5	5
2	5	5	5
3	5	5	5
4	5	5	5
5	5	max{4, 5} = 5	5
6	5	9	9
7	5	9	9
8	5	9	9
9	5	9	9
10	5	9	max(tab[2][0]+value[3], tab[2][10])=12
11	5	9	12 +5 = 13

子问题：容量为k的时候放前i件物品的最大总价值

base case：容量为0 =》0

放入第1件物品：

如果容量〉第一件物品重量：就是第一件物品价值

否则为0

recursion：

dp[i][j] = max(dp[i-1][j-weight[i]]+value[i],dp[i-1][j]) ({i,j|0<i<=n,0<=j<=total})

多重背包

例1 线性DP

There are N lily pads in a row. A frog starts on the leftmost lily pad and wishes to get to the rightmost one. The frog can only jump to the right. There are two kinds of jump the frog can make: • The frog can jump 3 lily pads to the right (skipping over two of them) • The frog can jump 5 lily pads to the right (skipping over four of them) Each lily pad has some number of flies on it. Design an algorithm that maximizes the total number of flies on lily pads the frog lands on getting to the rightmost lily pad.

到达最右边荷叶的时候得到的苍蝇的总数

=》 到达第k个荷叶的时候得到的苍蝇总数

Solution: Let us denote the number of flies on lily pad k by f(k). **Subproblem**(k): “Find the largest total number of flies N(k) on all lily pads the frog lands on while getting to the lily pad k.” If a lily pad k cannot be accessed from the first lily pad by jumps satisfying the prescribed constraints, let us define N(k) = −∞. **Recursion**: Clearly, N(1) = f(1); the immediately accessible lily pads (with a single jump) are lily pad 4 and lily pad 6. Thus N(2) = N(3) = N(5) = −∞ and N(4) = f(1) + f(4) and N(6) = f(1) + f(6).

For k > 6 the recursion formula is N(k) = max{N(k − 5), N(k − 3)} + f(k)

Note that if lily pad k − 5 and lily pad k − 3 are both inaccessible from lily pad 1, this recursion will correctly set the value of N(k) to −∞. **Solution of the original problem**: It is just the last obtained value N(n).

例2 （区间）

Given a sequence of n real numbers A1 . . . An, determine in linear time a contiguous subsequence Ai. . . Aj (i ≤ j) for which the sum of elements in the subsequence is maximized.

1，2，-3，4，7

max = 11

n = 5

k = 1

optSuffix(1) = 1

optSum(1) = 1

optSuffix(2) = 1

optSum(2) = 3

optSuffix(3) = 1

optSum(3) = 0

optSuffix(4) = 4

optSum(4) = 4

optSuffix(5) = 4

optSum(5) = 11

Solution:

• **Subproblem(k):** "find $m \leq k$ such that suffix $A_m A_{m+1} \dots A_k$ of the subsequence $A_1 A_2 \dots A_k$ has a maximal possible sum."
Note that such a suffix might consist of a single element A_k.

• **Recursion:** let OptSuffix(k) denote $m \leq k$ such that suffix $A_m A_{m+1} \dots A_k$ of the subsequence $A_1 A_2 \dots A_k$ has the largest possible sum and let this sum be denoted by OptSum(k). Then OptSuffix(1) = 1 and OptSum(1) = A_1 and for $k > 1$

$$\text{OptSuffix}(k) = \begin{cases} \text{OptSuffix}(k-1) & \text{if } \text{OptSum}(k-1) > 0; \\ k & \text{otherwise} \end{cases}$$

$$\text{OptSum}(k) = \begin{cases} \text{OptSum}(k-1) + A_k & \text{if } \text{OptSum}(k-1) > 0; \\ A_k & \text{otherwise} \end{cases}$$

Thus, if the sum OptSum($k-1$) of the optimal suffix $A_m A_{m+1} \dots A_{k-1}$ (where $m = \text{OptSuffix}(k-1)$) for the subsequence $A_1 A_2 \dots A_{k-1}$ is positive, then we extend such a suffix with A_k to obtain optimal suffix $A_m A_{m+1} \dots A_{k-1} A_k$ for the subsequence $A_1 A_2 \dots A_k$ and consequently OptSuffix(k) is still equal to m and OptSum(k) = OptSum($k-1$) + A_k; otherwise, if OptSum($k-1$) is negative we discard such optimal suffix and start a new one consisting of a single number A_k; thus, in this case OptSuffix(k) = k and OptSum(k) = A_k.

• **Solution of the original problem:** After we obtain OptSum(k) and OptSuffix(k) for all $1 \leq k \leq n$ we find $1 \leq q \leq n$ such that the corresponding OptSum(q) is the largest among all OptSum(k) for $1 \leq k \leq n$; let the corresponding OptSuffix(q) be equal to p; then the solution for the initial problem is the subsequence $A_p \dots A_q$.

例3 (背包变形)

You are given a set of n types of rectangular boxes, where the i^{th} box has height h_i, width w_i and depth d_i. You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Of course, you can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.

Solution:

Let us take 6 instances of each box, and rotate them in all possible ways to obtain the 6 possible bases. Note that each box has (in general) three different rectangular faces and each rectangular face can be rotated in two possible ways, depending which of the two different edges of its base is "horizontal". For simplicity, after that, we no longer allow boxes to be rotated. Note that now we have in total $6n$ many boxes.

We now note that if a box B_2 (with a fixed orientation) can be placed on top of a box B_1 then, since both sides of the base of B_2 must be smaller than the corresponding sides of the base of B_1, the surface area of the base of B_2 must be strictly smaller than the surface area of the base of B_1.

• **Producing the right ordering for recursion:** We order all boxes in a decreasing order of the surface area of their bases (putting boxes with the same surface area of their bases in an arbitrary order); then in every possible legitimate stack of boxes if a box $B_{i_{k+1}}$ is placed on top of the box B_{i_k}, then $i_{k+1} > i_k$. From now on we will assume that all the boxes are ordered in such a way.

• **Subproblem(k):** "Find a stack built from boxes $B_1, B_2, \ldots B_k$, $1 \leq k \leq 6n$, *which must end with box B_k* and which is of the largest possible total height".

• **Recursion:** Let us denote by MaxHeight(k) the height of the stack which solves Subproblem(k). Let also length(m), width(m) and height(m) denote the length, width and height of box B_m, respectively. Then

$$\text{MaxHeight}(k) = \max_{1 \leq m \leq k-1} \{\text{MaxHeight}(m) + \text{height}(k) \ : \ \text{length}(k) < \text{length}(m) \ \&\& \ \text{width}(k) < \text{width}(m)\}$$

We also define PreviousBox(k) to be the argument m for which the above maximum is achieved.

• **Solution of the original problem:** After all of these subproblems are solved for all $1 \leq k \leq n$, we pick the largest of MaxHeight(k) to obtain the height of the tallest stack of boxes and can reconstruct such a stack using pointers PreviousBox(k).

Max flow

(https://blog.csdn.net/tengweitw/article/details/17766133)

网络流G=(v, E)是一个有向图，其中每条边(u, v)均有一个非负的容量值，记为c(u, v) ≥ 0。如果(u, v) ∉ E则可以规定c(u, v) = 0。网络流中有两个特殊的顶点，即源点s和汇点t。

与网络流相关的一个概念是流。设G是一个流网络，其容量为c。设s为网络的源点，t为汇点，那么G的流是一个函数f：V×V →R，满足以下性质：

容量限制：对所有顶点对u，v∈V，满足f(u, v) ≤ c(u, v)；(流量小于最大容量)

反对称性：对所有顶点对u，v∈V，满足f(u, v) = - f(v, u)；（流入顶点的量等于流出顶点的量）

流守恒性：对所有顶点对u∈V-{s, t}，满足Σv∈V f(u,v)=0。

弧的类型:

a.饱和弧:即f(u,v)=c(u,v)；（容量等于流量）

b.非饱和弧:即f(u,v)<c(u,v)；（流量小于容量）

c.零流弧:即f(u,v)=0；（没有流量）

d.非零流弧:即f(u,v)>0.（流量大于0）

链:在容量网络中,称顶点序列(u1,u2,u3,u4,..,un,v)为一条链要求相邻的两个顶点之间有一条弧.（Path，walk）

设P是G中一条从Vs到Vt的链,约定从Vs指向Vt的方向为正方向.在链中并不要求所有的弧的方向都与链的方向相同.

a.前向弧:(方向与链的正方向一致的弧),其集合记为P+,

b.后向弧:(方向与链的正方向相反的弧),其集合记为P-.

Ford-Fulkerson算法是一种迭代算法

称作“扩充路径方法”，该方法是大量算法的基础，有多种实现方法。首先对图中所有顶点对的流大小清零，此时的网络流大小也为0。在每次迭代中，通过寻找一条“增广路径”(augument path)来增加流的值。**增广路径可以看作是源点s到汇点t的一条路径，并且沿着这条路径可以增加更多的流**。迭代直至无法再找到增广路径位置，此时必然从源点到汇点的所有路径中都至少有一条边的满边（即边的流的大小等于边的容量大小）。

其基本思想为：

给定一个流网络G和一个流，流的残留网Gf拥有与原网**相同的顶点**。原流网络中每条边将对应残留网中一条或者两条边，对于原流网络中的任意边(u, v)，流量为f(u, v)，容量为c(u, v)：

如果f(u, v) > 0，则在残留网中包含一条容量为f(u, v)的边(v, u);

如果f(u, v) < c(u, v)，则在残留网中包含一条容量为c(u, v) - f(u, v)的边(u, v)。

残留网允许我们使用任何广义图搜索算法来找一条增广路径，因为残留网中从源点s到汇点t的路径都直接对应着一条增广路径。

EK（Edmond—Karp）算法

反复寻找源点s到汇点t之间的增广路径，若有，找出增广路径上每一段[容量-流量]的最小值delta，若无，则结束。

在寻找增广路径时，可以用**BFS**来找，并且更新残留网络的值(涉及到反向边)。

而找到delta后，则使最大流值加上delta，更新为当前的最大流值。

例

You work for a new private university which wants to keep the sizes of classes small. Each class is assigned its maximal capacity - the largest number of students which can enroll in it. Students pay the same tuition fee for each class they get enrolled in. Students can apply to be enrolled in as many classes as they wish, but each of them will eventually be enrolled to at most 5 classes at any given semester. You are given the wish lists of all students, containing for each student the list of all classes they would like to enroll this particular semester and you have to chose from the classes they have put on their wish lists in which classes you will enroll them, without exceeding the maximal enrolment of any of the classes and without enrolling any student into more than 5 classes. Your goal is, surprisingly, to maximize the income from the tuition fees for your university. Design an efficient algorithm for such a task.

找vertex，容量，s and t。跑最大流算法得到答案

1. 学生作为a类点，课堂作为b类点
2. 设虚拟源点（**source**/super source）和汇点（sink/super sink）
3. 从s到每个a类点连线
4. b类点到t连线
5. ab之间根据学生的wish list连线，list上只要存在学生i和课程j的记录，我们就连起来
6. s到a类点连线的capacity是5
7. a类点到b类点的capacity是1（每个学生只能enrol同一门课一次）
8. b类点到汇点 capacity是对应的课程的最大容量
9. 跑最大流算法，得出的结果是学生注册课程的人次，乘上每门课的学费就是最大学费

Explain in detail why in a network the maximal flow from the source s to the sink t produced by adding augmenting paths using the Ford Fulkerson method is independent of the choice of augmenting paths.