动态规划

关键点：

```
1.状态表示
   1.1 集合dp[i]表示什么：
   1.2 属性是什么：个数、最大值
2.状态计算：状态方程
3.初始化

4.细节点：
    遍历顺序的重要性：先遍历背包还是先遍历物品
    打印dp数组
```

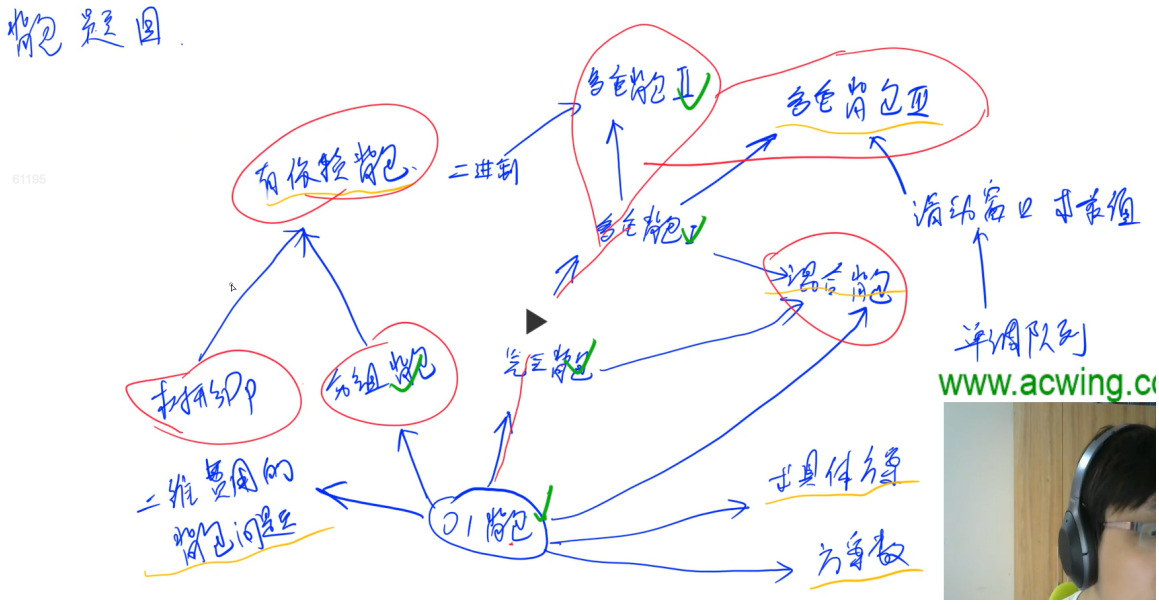
常见状态表示

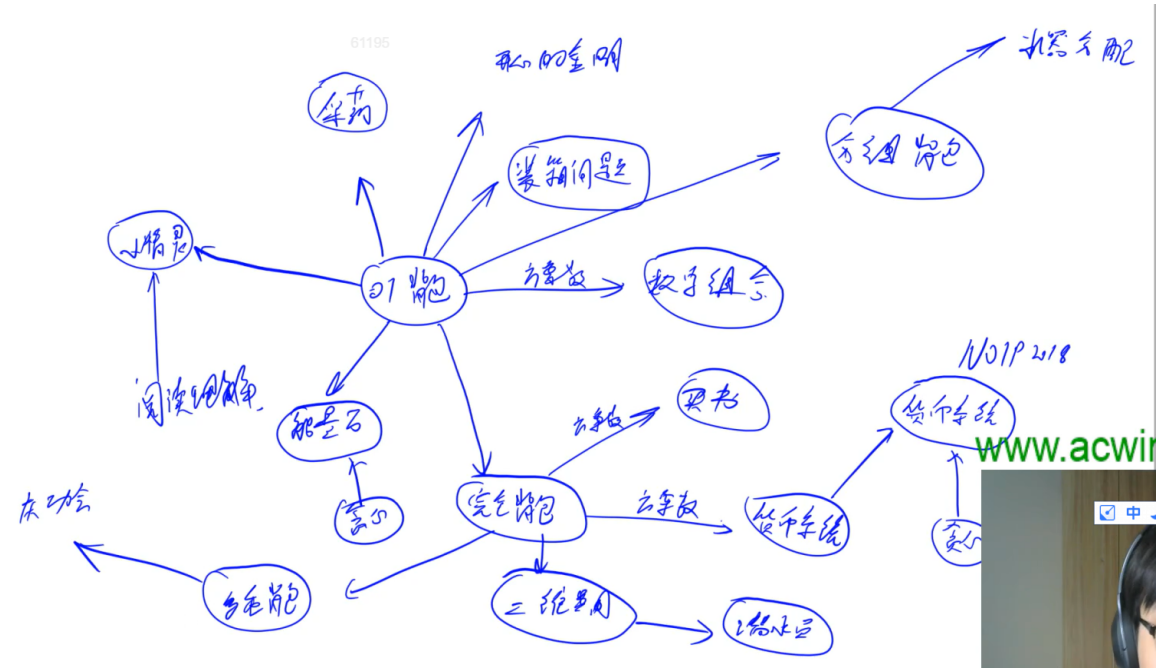
- `f[i][j]`表示前`i`个物品，体积不超过`j`的选法的最大值
- `f[i][j]`表示前`i`个数中选，和为`j`的选法的方案数
- `f[i][j][k]`表示前`i`个物品，限制`1`：不超过`j`，限制`2`：不超过`k`的选法
- `f[i][j]`表示总和为`i`，第一个数是`j`的选法序列的方案数
- `f[i][k]`，`k=0,1`。表示前`i`个数中选，第`i`个数的状态是`k`（选或不选）的选法集合的方案数
- `f[i][j][k]`，`k=0,1`。表示前`i`个数中选，限制：不超过`j`次，第`i`个数的状态是`k`（选或不选）的选法的方案数
- `f[u][k]`，`k=0,1`。表示以`u`为根节点的子树，根节点`u`选或不选的选法集合的属性
- `f[i]`表示以`arr[i]`结尾的子序列的属性
- `f[i][j]`表示`s1`从前`i`个字符中选，`s2`从前`j`个字符中选的方案数
- `f[i][j]`表示区间`[l,r]`内的合并问题的属性

1.背包模型

> 至少、恰好、至多的背包模型总结：https://www.acwing.com/blog/content/458/

- 所有背包模型的一维优化中，只有《完全背包的一维优化》和《多重背包的单调队列优化》这种情况体积的循环是从小到大，其他所有的情况 体积都是从大到小枚举
- 所有的二维情况都是体积和物品的循环顺序可以随意。





2.数字三角形模型

需要注意边界的问题（求最小值时么要初始化成INF）。下标尽量从1开始。这样才能保证使用滚动数组

3.最长上升子序列模型

> dp[i]表示以arr[i]结尾的一类问题

题目：

> lc300：最长上升子序列
>
> lc53.：连续子数组最大和
>
> lc940：不同的子序列 II（计算一个串的所有不相等子序列的个数）
>
> AcWing 1017. 怪盗基德的滑翔翼

4.前i前j模型

> dp [i] [j]表示从A的前i个字符中选择，从B的前j个字符 的方案数

> 最长公共子序列
>
> lc115. 不同的子序列(寻找A串中与B相等的子序列的个数)
>
> acwing902：最短编辑距离

树形dp

O（n），n是节点总数

- 根节点在dp数组中，`dp[u][1]`表示以u为根节点的，并且选择根节点的情况，

  例子：没有上司的舞会
- 计算树的向下最大高度

没有上司的舞会

https://www.acwing.com/activity/content/code/content/5382741/

```
    static void dfs(int root) {
        //dp[root][0] = 0;
        dp[root][1] = happy[root];
        for(int i = h[root];i!=-1;i = ne[i]){
            int j = e[i];
            dfs(j);//递归子树
            dp[root][0] += Math.max(dp[j][0],dp[j][1]);
            dp[root][1] += dp[j][0];
        }
    }
```

树的重心

给定一颗树。请你找到树的重心，并输出将重心删除后，剩余各个连通块中点数的最大值。

重心定义：重心是指树中的一个结点，如果将这个点删除后，剩余各个连通块中点数的最大值最小，那么这个节点被称为树的重心。

```
//返回节点u的所有子树的节点总个数
static int dfs(int u,int fa) {
    int sum=0,mx=0;
    for(int i=h[u];i!=-1;i=ne[i]) {
        int j = e[i];
        if(j==fa) continue;
        int cnt = dfs(j,u);
        mx = Math.max(mx,cnt);
        sum+=cnt;
    }
    mx = Math.max(mx,n-sum-1);
    if(mx < ans) {
        ans = mx;
```

```
            ansIdx = u;
        }
        return sum+1;
    }
```

向下的最长路径

dfs(int u)函数返回u作为根节点时，向下的最长路径。

```
    static int dfs(int u,int fa){
        int dist=0;
        for(int i=h[u];i!=-1;i=ne[i]){
            int j = e[i];
            if(j==fa) continue;
            int d = dfs(j,u) + w[i];
            dist = Math.max(dist,d);
        }
        return dist;
    }
```

树的直径

> AcWing 1072. 树的最长路径

计算向下的最长、第二长路径长度。

```
    static int dfs(int u,int fa) {
        int d1 = 0,d2 = 0;
        for(int i=h[u];i!=-1;i=ne[i]) {
            int j = e[i];
            if(j==fa) continue;
            int len = dfs(j,u)+w[i];
            if(len>=d1) {
                d2 = d1;
                d1 = len;
            }else if(len>d2) {
                d2 = len;
            }
        }
        ans = Math.max(ans,d1+d2);
        return d1;
    }
```

树的中心

给定一棵树，每条边都有一个权值。请你在树中找到一个点，使得该点到树中其他结点的最远距离最近。

https://www.acwing.com/activity/content/code/content/5390588/

```
    //计算节点向下的最长距离、次长距离
    //用子节点更新父节点（先递，在归的时候计算）
    static int dfs_down(int u,int fa) {
        int dd1 = 0,dd2 = 0;
        int dd1_point = 0;
```

```
        for(int i=h[u];i!=-1;i=ne[i]) {
            int j = e[i];
            if(j==fa) continue;
            int len = dfs_down(j,u)+w[i];
            if(len>=dd1) {
                dd2 = dd1;
                dd1 = len;
                dd1_point = j;
            }else if(len>dd2) {
                dd2 = len;
            }
        }
        d1[u] = dd1;
        d2[u] = dd2;
        p1[u] = dd1_point;
        return dd1;
    }
    //计算向上的最长距离
    //用父节点更新子节点（先计算，再递归）
    static void dfs_up(int u,int fa) {
        ans = Math.min(ans,Math.max(d1[u],up[u]));
        for(int i=h[u];i!=-1;i=ne[i]) {
            int j = e[i];
            if(j==fa) continue;
            up[j] = Math.max(up[u],p1[u]!=j?d1[u]:d2[u])+w[i];
            dfs_up(j,u);
        }
    }
```

区间dp

O（n^3）

- 普通区间dp

- 环形区间dp，在长度是2*n的数组中合并区间

```
for(int len=2;len<=n;len++) {
    for(int i=1;i+len-1<=n+n;i++) {
        int j = i+len-1;
        for(int k=i;k<j;k++) {
            dp[i][j] = Math.max(dp[i][j], dp[i][k] +dp[k+1][j] + sum[j]-sum[i-
1]);

        }
    }
}

//初始化
        for (int i = 1; i < N; i++) {//初始化到最大为N
            Arrays.fill(dp[i],-0x3f3f3f3f);
            dp[i][i] = 0;//一堆不需要合并，代价是0
        }
```

数位dp

- 题目形式：找一个区间内满足某个性质的数的个数，数据范围会开到10^10以上
- 数位dp思路：计算0~n中满足条件的个数
- 答案：count(b)-count(a-1)
- 难点：左子树计算公式（很多都用到了右边待枚举的数位个数i）

- 关键点
  - 1.找到任意一位上是哪些值对答案有贡献
  - 2.根据步骤1中找到的对答案有贡献的数位，根据这个对当前位进行划分
  - last存放的是什么
  - 左子树的公式能不能用dp
  - 3.最后的最右的右子树可能多一种答案。
  - i是从n-1枚举到0，当枚举到第i位时，**后面待枚举的数位的个数也是i**，这是非常重要的一个细节
  - 有时需要加上if(n==0) return 1;特判n==0是否满足条件

    如果不满足，这句加不加无所谓，因为不加这个返回值也是0

    但是如果满足，那么一定要return1。否则默认返回0，如果求1~5直接满足的个数，最后的结果就会多算一个。那么此时count(5)-count(0) = count(5)包括了0这个情况

```
static int count(int n){
    if(n==0) return 1;//判断0是否满足条件
    ArrayList<Integer> nums = new ArrayList<>();
    while(n>0){
        nums.add(n%10);
        n/=10;
    }
    n = nums.size();
    int res = 0;
    int last = 0;//存放上一时刻的状态，可以是上一位的值，也可以是前面的和
    for (int i = n-1; i >= 0; i--) {
        int x = nums.get(i);
        //左子树计算公式
        for (int j = 0; j < x; j++) {
        }
        //右子树一直递归

         // 右子树的终点可能会多一种方案
         if(i==0 && ) res++;
    }
    return res;
}
```