第四章-作业

1

在一条直线上有 n 堆石子，每堆有一定的数量，每次可以将两堆相邻的石子合并，合并后放在两堆的中间位置，合并的费用为两堆石子的总数。（30分）

求把所有石子合并成一堆的最小花费(定义 $dp[i][j]$ 为第 i 堆石子到第 j 堆合并的最小花费)。
(1) 写出该问题的递推方程。（10分）
(2) 有 5 堆石子 ($n=5$)，每堆石子大小分别为 $<1,3,5,2,4>$,求出把所有石子合并成一堆的最小花费(要求写出运算矩阵)。（10分）
(3) 写出该问题的伪代码。（10分）

答:

设 $w[i]$ 为第 i 堆石子的数量;

$R[i][j]$ 为第 i 堆石子到第 j 堆石子的总数量，即 $R[i][j]=\sum_{r=i}^{j} w[r]$.

(1)

$$dp[i][j]=\begin{cases}0, & \text{if } i=j;\\ min_{i\le k<j}\{dp[i][k]+dp[k+1,j]+R[i][j]\}, & \text{if } j>i.\end{cases}$$

(2)

$R[1][1]=1$	$R[1][2]=4$	$R[1][3]=9$	$R[1][4]=11$	$R[1][5]=15$
\	$R[2][2]=3$	$R[2][3]=8$	$R[2][4]=10$	$R[2][5]=14$
\	\	$R[3][3]=5$	$R[3][4]=7$	$R[3][5]=11$
\	\	\	$R[4][4]=2$	$R[4][5]=6$
\	\	\	\	$R[5][5]=4$

$dp[1][1]=0$	$dp[1][2]=4$	$dp[1][3]=13$	$dp[1][4]=22$	$dp[1][5]=34$
\	$dp[2][2]=0$	$dp[2][3]=8$	$dp[2][4]=17$	$dp[2][5]=28$
\	\	$dp[3][3]=0$	$dp[3][4]=7$	$dp[3][5]=17$
\	\	\	$dp[4][4]=0$	$dp[4][5]=6$
\	\	\	\	$dp[5][5]=0$

(3)

$stone\_merge(n,w[])$

```
for i <- 1 to n do
    dp[i][i] <- 0
    R[i][i] <- w[i]
for l <- 2 to n do
    for i <- 1 to n - l + 1 do
        j = i + l - 1
        dp[i][j] = MAX_NUM
        R[i][j] = R[i][j - 1] + w[j]
        for k <- i to j - 1 do
            q = dp[i][k] + dp[k + 1][j] + R[i][j]
            if(q < dp[i][j]) then
                dp[i][j] = q
return dp[1][n]
```

c 语言代码：1.c

2

若 7 个关键字的概率如下所示，求其最优二叉搜索树的结构和代价，要求必须写出递推方程。（30分）

i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.02	0.10	0.12	0.14
q_j	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

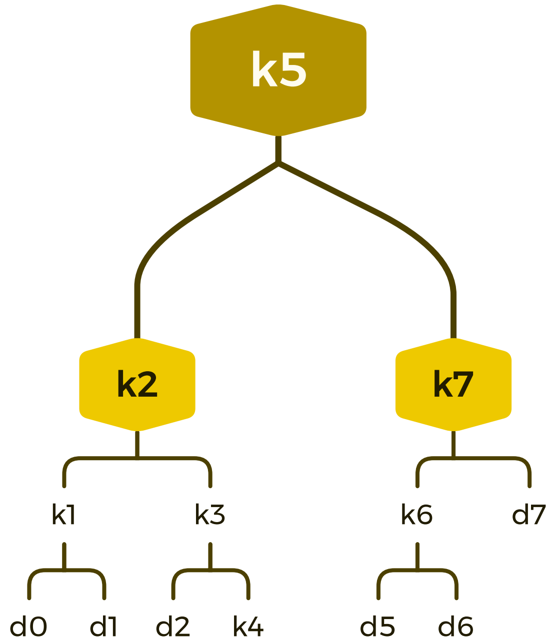
答:

递推方程

$$E(i,j)=\begin{cases} q_{i-1}=q_j, & \text{if } j=i-1; \\ min_{i\le k\le j}\{E(i,R-1)+E(r+1,j)+W(i,j)\}, & \text{if } j>i. \end{cases}$$

$$W(i,j)=\sum_{l=i}^{j} p_l+\sum_{l=i-1}^{j} q_l$$

最优二叉搜索树的结构



代价为 3.12.

3

编程题：兑换零钱问题（40分）

题目描述：

给定不同面额的硬币 $coins$ 和一个总金额 $amount$。编写一个函数来计算可以凑成总金额所需的最少的硬币个数。如果没有任何一种硬币组合能组成总金额，返回-1。（提示：你可以认为每种硬币的数量是无限的）。

示例 1:

> 输入: coins = [1, 2, 5], amount = 11
>
> 输出: 3
>
> 解释: 11 = 5 + 5 + 1

示例 2:

> 输入: coins = [2], amount = 3
>
> 输出: -1

运用动态规划的思想作答，请写出分析过程和状态转移方程，并用一种语言（最好是 $C++$ 或 $JAVA$）实现你的思路，并保证代码能正确运行，复杂度尽可能低。

答：

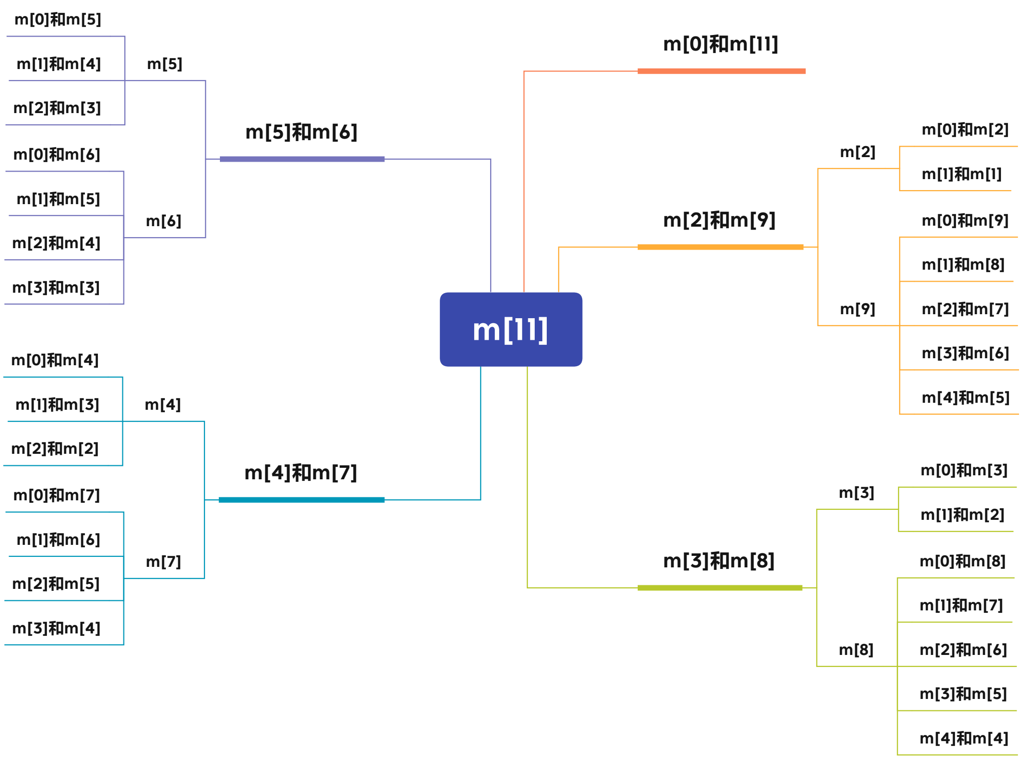
1. 分析优化解的结构

   设凑成总金额 i 所需的最少的硬币个数为 $m[i]$。

若计算 $m[i]$ 的优化顺序在 k 处分开，即 $m[i]=m[k]+m[n-k]$，则在 $m[i]$ 的优化顺序中，对应于子问题 $m[k]$ 的解必定是 $m[k]$ 的优化解；对应于子问题 $m[n-k]$ 的解必须是 $m[n-k]$ 的优化解。

证明：反证法

子问题的重叠性：以上示例 1 为例，可见重复的 $m[i]$ 非常多。



2. 递归地定义最优值的代价

$$m[i]=\begin{cases}0, & \text{if } i=0;\\ 1, & \text{if } i\in costs[];\\ min_{0\le k\le n/2}\{m[k]+m[n-k]\}, & \text{if 其他}.\end{cases}$$

3. 自底向上地计算优化解的代价保存之，并获取构造最优值的信息 & 根据构造最优值的信息构造优化解

   一维数组，由小到大计算即可。

4. 源代码

```
#include <stdio.h>
const int MAX_NUM = 100000;
int min(int a, int b)
{
    if (a > b)
        return b;
    else
        return a;
}
int mininum(int coins[], int n, int amount)
{
    int m[amount + 2];
    for (int i = 1; i <= amount; i++)
        m[i] = MAX_NUM;

```

```c
    m[0] = 0;
    for (int i = 1; i <= n; i++)
        m[coins[i]] = 1;
    for (int i = 1; i <= amount; i++)
        for (int k = 0; k <= i / 2; k++)
            m[i] = min(m[i], m[k] + m[i - k]);

    if (m[amount] == MAX_NUM)
        return -1;
    else
        return m[amount];
}
int main()
{
    int coins1[4] = {0, 1, 2, 5};
    int amount1 = 11;
    int coins2[2] = {2};
    int amount2 = 3;
    printf("%d\n", mininum(coins1, 3, amount1));
    printf("%d\n", mininum(coins2, 1, amount2));
    return 0;
}
```

源文件：3.c