学号：ZY2206117　姓名：**黄海浪**　作业：Assignment_2

题目 1:

1. 符号说明

需求量n_k：表示第k月的产品需求量。
状态变量s_k：表示第k月的产品库存量。
决策变量x_k：表示第k月产品生产数量。
状态转移方程：$s_{k+1} = s_k + x_k - n_k$
允许决策集合$D_k(x_k)$：$D_k(x_k) = \{\max(n_k - s_k, 0) \le x_k \le 6\}$
成本v_k：表示第k月的产生成本。
最优值函数$f_k(s_k)$：表示第k月到第n月采取最优策略产生的成本最小值。

2. 递推关系式

$$f_k(s_k) = \min\{v_k + f_{k+1}(s_{k+1})\}$$

其中：

$$v_k = \begin{cases} 0.5 * (s_k - n_k)\,, x_k = 0 \\ 3 + x_k + 0.5 * (s_k + x_k - n_k)\,, x_k \ne 0 \end{cases}$$

$f_5(s_5) = 0$且$k \in [1,5]$

3. 计算步骤

（1）　当$k = 4$时, 由题$n_4 = 4$，$\max(s_4) = 4$且$s_5 = 0$，$D_4(x_4) = 4 - s_4$

s_4	x_4	s_5	x_5	$f_5(s_5)$	v_4	$v_4 + f_5(s_5)$
0	4	0	0	0	7	7
1	3	0	0	0	6	6
2	2	0	0	0	5	5
3	1	0	0	0	4	4
4	0	0	0	0	0	0

（2）　当$k = 3$时, 由题$n_3 = 2, D_3(x_3) = \{\max(2 - s_3, 0) \le x_3 \le 6\}$

s_3	x_3	s_4	x_4	$f_4(s_4)$	v_3	$v_3 + f_4(s_4)$
0	2	0	4	7	5	12
	3	1	3	6	6.5	12.5
	4	2	2	5	8	13
	5	3	1	4	9.5	13.5
	6	**4**	**0**	**0**	**11**	**11**
1	1	0	4	7	4	11
	2	1	3	6	5.5	11.5
	3	2	2	5	7	12
	4	3	1	4	8.5	12.5
	5	**4**	**0**	**0**	**10**	**10**
2	**0**	**0**	**4**	**7**	**0**	**7**
	1	1	3	6	4.5	10.5
	2	2	2	5	6	11

	3	3	1	4	7.5	11.5
	4	4	0	0	9	9
3	**0**	**1**	**3**	**6**	**0.5**	**6.5**
	1	2	2	5	5	10
	2	3	1	4	6.5	10.5
	3	4	0	0	8	8
4	**0**	**2**	**2**	**5**	**1**	**6**
	1	3	1	4	5.5	9.5
	2	4	0	0	7	7
5	**0**	**3**	**1**	**4**	**1.5**	**5.5**
	1	4	0	0	6	6
6	**0**	**4**	**0**	**0**	**2**	**2**

因此，$k=3$时的最优决策如下：

s_3	x_3	s_4	x_4	$f_4(s_4)$	v_3	$v_3+f_4(s_4)$
0	6	4	0	0	11	11
1	5	4	0	0	10	10
2	0	0	4	7	0	7
3	0	1	3	6	0.5	6.5
4	0	2	2	5	1	6
5	0	3	1	4	1.5	5.5
6	0	4	0	0	2	2

（3） 当$k=2$时, 由题$n_2=3, D_2(x_2)=\{\max(3-s_2,0)\le x_2\le 6\}$

s_2	x_2	s_3	x_3	$f_3(s_3)$	v_2	$v_2+f_3(s_3)$
0	3	0	6	11	6	17
	4	1	5	10	7.5	17.5
	5	**2**	**0**	**7**	**9**	**16**
	6	3	0	6.5	10.5	17
1	2	0	6	11	5	16
	3	1	5	10	6.5	16.5
	4	**2**	**0**	**7**	**8**	**15**
	5	3	0	6.5	9.5	16
	6	4	0	6	11	17
2	1	0	6	11	4	15
	2	1	5	10	5.5	15.5
	3	**2**	**0**	**7**	**7**	**14**
	4	3	0	6.5	8.5	15
	5	4	0	6	10	16
	6	5	0	5.5	11.5	17
3	**0**	**0**	**6**	**11**	**0**	**11**
	1	1	5	10	4.5	14.5

	2	2	0	7	6	13
	3	3	0	6.5	7.5	14
	4	4	0	6	9	15
	5	5	0	5.5	10.5	16
	6	6	0	2	12	14
4	**0**	**1**	**5**	**10**	**0.5**	**10.5**
	1	2	0	7	5	12
	2	3	0	6.5	6.5	13
	3	4	0	6	8	14
	4	5	0	5.5	9.5	15
	5	6	0	2	11	13
5	**0**	**2**	**0**	**7**	**1**	**8**
	1	3	0	6.5	5.5	12
	2	4	0	6	7	13
	3	5	0	5.5	8.5	14
	4	6	0	2	10	12
6	**0**	**3**	**0**	**6.5**	**1.5**	**8**
	1	4	0	6	6	12
	2	5	0	5.5	7.5	13
	3	6	0	2	9	11

因此，$k=2$时的最优决策如下：

s_2	x_2	s_3	x_3	$f_3(s_3)$	v_2	$v_2+f_3(s_3)$
0	5	2	0	7	9	16
1	4	2	0	7	8	15
2	3	2	0	7	7	14
3	0	0	6	11	0	11
4	0	1	5	10	0.5	10.5
5	0	2	0	7	1	8
6	0	3	0	6.5	1.5	8

（4） 当$k=1$时, 由题$s_1=0$，$n_1=2, D_1(x_1)=\{2\le x_1\le 6\}$

s_1	x_1	s_2	x_2	$f_2(s_2)$	v_1	$v_1+f_2(s_2)$
0	2	0	5	16	5	21
	3	1	4	15	6.5	21.5
	4	2	3	14	8	22
	5	**3**	**0**	**11**	**9.5**	**20.5**
	6	4	0	10.5	11	21.5

因此，决策$P_{1,n}=\{x_1,x_2,x_3,x_4\}=\{5,0,6,0\}$可以达到最低成本，最低成本为20.5。即 1 月生产 5 单位，3 月生产 6 单位。

题目 2：

1. 符号说明

阶段k：表示当前已遍历过k个节点，$k=1$表示从v_1出发，$k=7$表示回到v_1。
状态变量$s_k=(i,N_k)$：表示第k阶段的目前所在节点i和剩余未遍历的节点N_k。
决策变量$x_k=(i,j)$：表示第k阶段的目前所在节点i和下一个遍历的点。
状态转移方程：$s_{k+1}=(s_k\to x_k)=(j,N_{k+1})=(j,N_k-\{j\})$
允许决策集合$d_k(x_k)$：$d_k(x_k)=\{x_k(j)\in N_k\}$
成本$c_k=D(i,j)$：表示第k阶段产生的花费。
最优值函数$f_k(s_k)=f_k(i,N_k)$：从i节点经过N_k节点1次且仅1次返回v_1采取最优策略产生的最小成本。

2. 递推关系式

$$f_k(s_k)=\min\{c_k+f_{k+1}(s_{k+1})\}=\min\{D(i,j)+f_{k+1}(j,\{N_k-j\})\}$$

$f_7(s_7)=0$且$k\in[1,7]$

3. 程序伪代码

首先构建 dp 表表示从城市 i 出发经过城市 j 到达城市 1 的最小距离，dp 表如下（假设有 4 个城市）：

索引	0	1	2	3	4	5	6	7
	000	001	010	011	100	101	110	111
	{Ø}	{1}	{2}	{1,2}	{3}	{1,3}	{2,3}	{1,2,3}
0								
1								
2								
3								

对于第x个城市，它的二进制可以表示为$1\ll(x-1)$，即第x位为 1。同样，判断某个索引是否包含某个城市可以用$x\gg(i-1)\&1$判断。

算法为代码 TSP：

```
TSP(D,n)
//旅行商问题动态规划求解
//输入：邻接矩阵花费 D 和节点个数 n
//输出：无，但会更新 dp 表

//init 即各个城市直接到城市 1 的距离
for 城市 i in 所有的城市
    dp[i][0] = D[i][0]
end

//dp
for j in 所有的城市组合
    for i in 所有的城市
        if i 不在 j 中
            for k in j 中城市
                // 更新 i 经过 j 的距离，通过与 i-k 再经过 j-{k}的距离比较
                dp[i][j] = min(dp[i][j], d[k][j-{k}]+D[i][k])
```

```
            end
        end
    end
end
```

算法为代码 getPath：

```
getPath()
// 根据 dp 表的记录获得路径
// 输入：无
// 输出：无（打印路径）

i = 0
while(j 中城市不为空) do
    打印 城市 i
    for k in j 中所有城市
        // 如果最优值通过 k 城市计算而来
        if dp[i][j] == dp[i][j-{k}]+D[i][k]
            i = k
            j = j-{k}
            break
        end
    end
end
打印 城市 i
```

4. 程序相关说明、结果和分析（更多请参考代码文件：main.cpp）

运行环境：

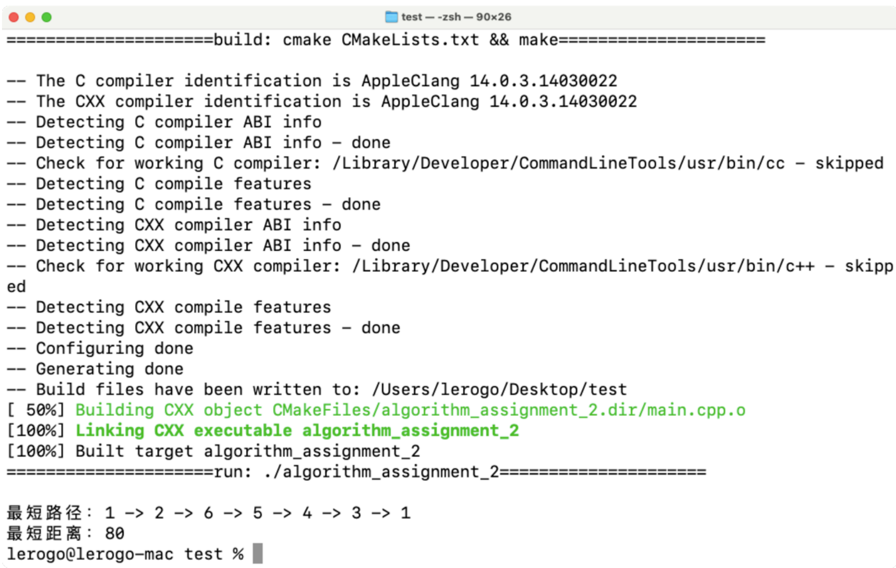
- 操作系统：macOS Ventura13.3.1
- 构建工具：cmake version 3.25.2；GNU Make 3.81
- 构建命令：cmake CMakeLists.txt && make
- 运行命令：./algorithm_assignment_2

运行结果：

最短路径：1 -> 2 -> 6 -> 5 -> 4 -> 3 -> 1

最短距离：80

结果截图：



```
test — -zsh — 90×26
====================build: cmake CMakeLists.txt && make====================

-- The C compiler identification is AppleClang 14.0.3.14030022
-- The CXX compiler identification is AppleClang 14.0.3.14030022
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /Library/Developer/CommandLineTools/usr/bin/c++ - skipp
ed
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/lerogo/Desktop/test
[ 50%] Building CXX object CMakeFiles/algorithm_assignment_2.dir/main.cpp.o
[100%] Linking CXX executable algorithm_assignment_2
[100%] Built target algorithm_assignment_2
====================run: ./algorithm_assignment_2====================

最短路径: 1 -> 2 -> 6 -> 5 -> 4 -> 3 -> 1
最短距离: 80
lerogo@lerogo-mac test %
```

5. 附录-代码

```cpp
/*
 * encoding: utf-8
 * */

#include <iostream>
#include <limits.h>

// 6 cities
const int n = 6;
const int m = 1 << (n - 1);
// 6x6 cost matrix
const int D[n][n] = {
        {0,  10, 20, 30, 40, 50},
        {12, 0,  18, 30, 25, 21},
        {23, 19, 0,  5,  10, 15},
        {34, 32, 4,  0,  8,  6},
        {45, 27, 11, 10, 0,  18},
        {56, 22, 16, 20, 12, 0}
};
// 表示从 i 出发经过 j 到城市 1 的距离
int dp[n][m];


void TSP() {
    // init 所有城市到城市 1 的距离
    for (int i = 0; i < n; i++) {
        dp[i][0] = D[i][0];
    }

    // dp
    for (int j = 1; j < m; j++) {
        for (int i = 0; i < n; i++) {
            dp[i][j] = INT_MAX;
            // if i is not in j 判断 i 是否在 j 中
            // j 是用二进制表示的，j 的第 i 位为 1 表示 i 在 j 中
            if ((j >> (i - 1)) & 1) {
                continue;
            }
            // 遍历 j 中的每个城市 k，找到最小的 dp[i][j]
            for (int k = 1; k < n; k++) {
                if ((j >> (k - 1)) & 1) {
                    // j - {k} 表示 j 中除了 k 之外的城市
                    // 由于 j 中包含 k，所以 j - {k}中的城市数比 j 中的城市数少 1
                    // 所以 dp[k][j - {k}]已经在上一次循环中计算过了
```

```
                dp[i][j] = std::min(dp[i][j], dp[k][j - (1 << (k - 1))] + D[i][k]);
            }
        }
    }
  }
}

void getPath() {
    // j = m - 1 表示所有城市都已经遍历过了
    // i = 0 表示从城市 1 开始
    int j = m - 1;
    int i = 0;
    while (j > 0) {
        // 从城市 1 开始，依次输出城市的编号
        std::cout << i + 1 << " -> ";
        for (int k = 1; k < n; k++) {
            if ((j >> (k - 1)) & 1) {
                // 如果 dp[i][j] == dp[k][j - {k}] + D[i][k]
                // 说明从 i 出发经过 j 到城市 1 的距离等于从 k 出发经过 j - {k}到城市 1 的距离加上 i 到 k 的
距离
                // 所以 i 到 k 是最短路径的一部分
                // 所以下一次循环应该从 k 开始
                if (dp[i][j] == dp[k][j - (1 << (k - 1))] + D[i][k]) {
                    // j -= (1 << (k - 1)) 表示 j 中去掉 k
                    j -= (1 << (k - 1));
                    i = k;
                    break;
                }
            }
        }
    }
    std::cout << i + 1 << " -> 1" << std::endl;
}

int main() {
    TSP();
    std::cout << "最短路径：";
    getPath();
    std::cout << "最短距离：";
    std::cout << dp[0][m - 1] << std::endl;
    return 0;
}
```