1. 作业 1

变量表示

- R_k 为第 k 个月初的库存剩余量
- S_k 为第 k 个月的生产量
- D_k 为第 k 个月的需求量
- W_k 为第 k 个月的生产成本

$$W_k = \begin{cases} 0, & if\ S_k = 0 \\ S_k + 3, & otherwise \end{cases}$$

- $C(k, R_k)$ 为从第 k 个月开始，且第 k 个月初库存剩余 R_k 的情况下，到第 4 个月末的总生产成本

状态转移方程为：

$$R_{k+1} = R_k + S_k - D_k$$

递推公式为：

$$C(k, R_k) = \min_{0 \le S_k \le 6} C(k+1, R_{k+1}) + W_k + 0.5R_k$$

其中，$R_{k+1} = R_k + S_k - D_k \ge 0$，且 $R_5 = 0, R_1 = 0$

详细计算过程如下：

- $k = 4$，$R_5 = R_4 + S_4 - D_4 = R_4 + S_4 - 4 = 0$

	0	1	2	3	4	5	6
c(4,0)					7		
c(4,1)				6.5			
c(4,2)			6				
c(4,3)		5.5					
c(4,4)	2						

- $k = 3$, $R_4 = R_3 + S_3 - D_3 = R_3 + S_3 - 2$

	0	1	2	3	4	5	6	min	S
c(3,0)			12	12.5	13	13.5	11	11	6
c(3,1)		11.5	12	12.5	13	10.5		10.5	5
c(3,2)	8	11.5	12	12.5	10			8	0
c(3,3)	8	11.5	12	9.5				8	0
c(3,4)	8	11.5	9					8	0
c(3,5)	8	8.5						8	0
c(3,6)	5							5	0

- $k=2,\ R_3=R_2+S_2-D_2=R_2+S_2-3, R_2\le 4$

	0	1	2	3	4	5	6	min	S
c(2,0)				17	17.5	16	17	16	5
c(2,1)			16.5	17	15.5	16.5	17.5	15.5	4
c(2,2)		16	16.5	15	16	17	18	15	3
c(2,3)	12.5	16	14.5	15.5	16.5	17.5	15.5	12.5	0
c(2,4)	12.5	14	15	16	17	15		12.5	0

- $k=1,\ R_2=R_1+S_1-D_1=R_1+S_1-1=S_1-2, S_1\ge 2$

	0	1	2	3	4	5	6	min	S
c(1,0)			21	21.5	22	20.5	21.5	20.5	5

所以，最低总成本为 20.5，对应 1-4 月的生产量为 5、0、6、0

2. 作业 2

变量表示

- $d[i][j]$ 表示从城市 v_i 到城市 v_j 的距离
- n 表示城市个数
- S 表示城市节点的集合，$S\subseteq\{v_1,\ldots,v_n\}$
- $D(S,j)$ 表示从 v_1 出发经过 S 中所有城市到达 v_j 的最短距离

递推关系式

$$D(S,j)=\min_{i\in S,i\neq j} D(S-\{j\},i)+d[i][j]$$

伪代码

Algorithm 1 计算从 v_1 出发经过其他所有城市仅一次回到 v_1 的最短距离

INPUT: 所有城市之间的距离表 d

OUTPUT: $D[2^n][n]$ 记录所有最短距离的中间状态, $choose[2^n][n]$ 记录最短距离对应的转移决策, $path[n]$ 记录最短路径

```
for all i, j ∈ [0, n), initial D[i][j] to −1
D[1][0] ← 0
for s = 1 → 2^n − 1 do
  for i = 0 → n − 1 do
    if D[s][i] = −1 then
      continue
    end if
    for j = 1 → n − 1 do
      if s ∧ (2^j) ≠ 0 then
        continue
      end if
      s_new = s ∨ (2^j)
      if D[s_new][j] = −1 ∨ D[s_new][j] > D[s][i] + d[i][j] then
        D[s_new][j] = D[s][i] + d[i][j]
        choose[s_new][j] = i
      end if
    end for
  end for
end for
result ← ∞
last = −1
for i = 1 → n − 1 do
  if D[2^n − 1][i] > 0 then
    if result > D[2^n − 1][i] + d[i][0] then
      result = D[2^n − 1][i] + d[i][0]
      last = i
    end if
  end if
end for
path[n − 1] = last
S = 2^n − 1
for i = n − 1 → 1 do
  if S ≤ 0 then
    break
  end if
  path[i − 1] = choose[S][path[i]]
  S = S − 2^path[i]
end for
return result, path
```

时间复杂度分析:
根据上述伪代码可以看出，城市集合有 2^n 种可能，对于每个城市集合，最大需要 n^2，故复杂度为 $O(n^2 2^n)$

c++ 代码:

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

using namespace std;
// number of cities
#define N  6
#define N_S (1<<N)

void tsp(int d[][N])
{
        int D[N_S][N];
        int choose[N_S][N];

        // initialize D
        for (int i = 0; i < N_S; i++) {
                for (int j = 0; j < N; j++) {
                        D[i][j] = -1;
                }
        }
        //put v1 to S
        D[1][0] = 0;
        for (int s = 1; s < N_S; s++) {
                for (int i = 0; i < N; i++) {
                        // i not in S
                        if (D[s][i] == -1)
                                continue;
                        // path from i to j
                        for (int j = 1; j < N; j++) {
                                // j in S
                                if ((s & (1<<j)) != 0)
                                        continue;
                                int s_new = s | (1<<j);
                                if (D[s_new][j] == -1 || D[s_new][j] >= D[s][i] + d[i][j]) {
                                        D[s_new][j] = D[s][i] + d[i][j];
                                        choose[s_new][j] = i;
                                }
                        }
                }
        }

        // from last city to v1
        int shortest_dis = INT_MAX;
        int last_city = -1;
        for (int i = 1; i < N; i++) {
                if (D[N_S-1][i] > 0) {
                        if (D[N_S-1][i] + d[i][0] < shortest_dis) {
                                shortest_dis = D[N_S-1][i] + d[i][0];
                                last_city = i;
                        }
                }
        }
```

```
        // recover path
        int path[N];
        path[N-1] = last_city;
        for (int i = N-1, s=N_S-1; i > 0 && s > 0; i--) {
                path[i-1] = choose[s][path[i]];
                s = s - (1<<path[i]);
        }

        // print path and shortest path
        cout << "Path:";
        for (int i = 0; i < N; i++) {
                cout << path[i]+1 << "->";
        }
        cout << "1" << endl;
        cout << "Distance:" << shortest_dis << endl;
}

int main()
{
        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, 16,
                       45, 27, 11, 10,  0, 18,
                       56, 22, 16, 20, 12, 0};

        // dp solve tsp
        tsp(d);
        return 0;
}
```

运行代码得：最短路径为 1→2→6→5→4→3→1 ，对应的最短距离为：80