L2C1 Lesson1

> 算法题分析过程：
>
> - 题目
>   - （阅读理解得到）模型
>   - 题目1
>   - 题目2
>   - 题目3
>   - 题目4
>   - ....
> - 算法上坐标系一般向下为x轴，向上为y轴，数学中相反

> 数字三角形模型
>
> - 很重要的常用的划分依据：“**最后**，不重（求max/min时可放松），不漏”
> - 摘花生
>   - f[i, j]映射的集合：最后一步是从上面下来，最后一步是从左边下来
>
>   状态计算：Max(f[i-1, j]+w[i, j], f[i, j-1]+w[i, j])
>
> ```
> #include <iostream>
> #include <algorithm>
>
> using namespace std;
>
> const int N = 110;
>
> int n, m;
> int w[N][N];
> int f[N][N];
>
> int main() {
>     int T;
>     scanf("%d", &T);
>     while (T--) {
>         scanf("%d%d", &n, &m);
>         for (int i=1; i<=n; i++)
>             for (int j=1; j<=m; j++) scanf("%d", &w[i][j]);
>
>         for (int i=1; i<=n; i++)     //注意下标起始位置
>             for (int j=1; j<=m; j++)
>                 f[i][j] = max(f[i-1][j], f[i][j-1])+w[i][j];
>
>         printf("%d\n", f[n][m]);
>     }
>
>     return 0;
> }
>
> ```
>
> - 最低通行费（摘花生变种，2n-1 -> 不能走回头路，min）
>
> ```
> #include <iostream>
> #include <algorithm>
>
> using namespace std;
>
> const int N = 110, INF = 2e9;
>
> int n;
> int w[N][N];
> int f[N][N];
>
> int main() {
>     scanf("%d", &n);
>
>     for (int i=1; i<=n; i++)
>         for (int j=1; j<=n; j++) scanf("%d", &w[i][j]);
>
>     for (int i=1; i<=n; i++)
>         for (int j=1; j<=n; j++) {
>             if (i==1 && j==1) f[i][j] = w[i][j]; //特判左上角
> ```

```
            else {
                f[i][j] = INF;
                if (i > 1) f[i][j] = min(f[i][j], f[i-1][j]+w[i][j]);   //i>1时可从上往下
                if (j > 1) f[i][j] = min(f[i][j], f[i][j-1]+w[i][j]);   //j>1时可从左往右
            }
        }

    printf("%d\n", f[n][n]);

    return 0;
}

```

- 方格取数（走两次）
  - 走两次但同样一个数只能被取一次
  - 状态表示：f[i1, j1, i2, j2]表示所有从(1,1), (1,1)分别走到(i1,j1), (i2, j2)的路径的最大值
  - 如何处理"同一个格子不能被重复选择"？（同时走）
    - 什么时候两条并行线最终会有交集？

      两条路线走过的总步数一致，即只有再i1+j1 == i2+j2 时（必要条件），两条路径的最终格子**才有可能**重合，以此优化状态表示。

      f[k, i1, i2]，其中k表示两条路线当前走到位置的横纵坐标之和（k=i1+j1=i2+j2），进一步表示所有从(1,1), (1,1)分别走到(i1,k-i1), (i2, k-i2)的路径**且格子只被取一次的**最大值
    - 怎么考虑状态计算？
      - 第一条路线最后一步下，第二条路线最后一步下
      - 第一条路线最后一步下，第二条路线最后一步右
      - 第一条路线最后一步右，第二条路线最后一步下
      - 第一条路线最后一步右，第二条路线最后一步右

      进一步分析
      - 第一类最大值：
        - (1, 1) -> (i1-1, j1) -> (i1, j1)
        - (1, 1) -> (i2-1, j2) -> (i2, j2)
        - 即第一类最大值为Max(f[k-1, i1-1, i2-1]) + (最后一步是否重合？w[i, j]: w[i1, j1]+w[i2, j2])
      - 同理分析第二三四类最大值

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 15;

int n;
int w[N][N];
int f[N*2][N][N];   //f[k][i1][i2], k=i1+j1=i2+j2

int main() {
    scanf("%d", &n);

    int a, b, c;
    while (cin >> a >> b >> c, a || b || c) w[a][b] = c;

    for (int k=2; k<=n*2; k++)
        for (int i1=1; i1<=n; i1++)
            for (int i2=1; i2<=n; i2++) {
                int j1 = k-i1, j2= k-i2;
                if (j1>=1 && j1<=n && j2>=1 && j2<=n) { //判断j的合法性
                    int t = w[i1][j1];
                    if (i1 != i2) t += w[i2][j2];   //最终格子不重合
                    int &x = f[k][i1][i2];  //通过引用简化代码
                    x = max(x, f[k-1][i1-1][i2-1]+t);   //第一类
                    x = max(x, f[k-1][i1-1][i2]+t);     //第二类
                    x = max(x, f[k-1][i1][i2-1]+t);     //第三类
                    x = max(x, f[k-1][i1][i2]+t);       //第四类
                }
            }

    printf("%d\n", f[n+n][n][n]);

    return 0;
}
```

- k取方格数（最小费用流）
  - DP是图论的子集，DP=>最短路问题，当最短路问题是拓扑图时，可反向转换