周赛 9 T1

简单的数一数每一种形状的 `L` 即可。（送分题）

有以下三种方式:

1. 外层形状，内层循环找。
2. 外层循环，内层找四种形状。
3. （我的做法）基于 2， 观察到一个 2 * 2 中有 3 个 1 等价于有 1 个 L，4 个 1 等价于有 2 个 L。

题目难(坑)点(?):

- 包含且仅包含三个 1 不代表四个 1 不行，对于不确定的题意，请看样例来解释。
- 输入不能用 int 来存，输入的是单个字符，应该用 char 来存储，再转化为 int。

```
#include <iostream>
using namespace std;

const int N = 200;
char a[N][N];

int main(){
    int n;
    cin >> n;
    for (int i = 0 ; i < n ; ++i)
        for (int j = 0 ; j < n ; ++j) {
            char c;
            cin >> c;
            a[i][j] = c - '0';
        }
    int ans = 0;
    for (int i = 1 ; i < n ; ++i)
        for (int j = 1 ; j < n ; ++j) {
            int sum =
                a[i - 1][j - 1] + a[i][j]
              + a[i - 1][j] + a[i][j - 1];
            if      (sum == 3) ans += 1;
            else if (sum == 4) ans += 4;
        }
    cout << ans;
    return 0;
}
```

周赛 9 T2

难度也不大，这里简单严谨证明一下正确性。看起来 $ax+by+cz$ 需要枚举 x,y,z 即三重循环，但实际上观察后容易发现，如果 $k=ax+by+cz$，且 $k\neq 0$，那么 x,y,z 中至少有一个是非 0 的。因此，不妨假设是 $x>0$，那么此时，我们容易发现 $k'=a(x-1)+by+cz=k-a$ 也在其中。因此，我们可以用递推的思路来解决。如果 $k\neq 0$ 可以表示，那么 $k-a$ 或 $k-b$ 或 $k-c$ 至少一个可以被表示，我们用一个数组维护当前数字是否可以被表示，然后枚举到每个数字的时候往前检查即可。由于枚举到当前数字的时候，前面的每个数字是否可表示都已经被处理掉了，所以正确性没有问题。

```
#include <iostream>
```

```cpp
#include <iostream>
using namespace std;

const int N = 1e6 + 3;
bool vis[N];

int main(){
    int T;
    cin >> T;
    vis[0] = 1; // 0 显然是可表示的
    while (T--) {
        int n,a,b,c;
        cin >> n >> a >> b >> c;
        int cnt = 0;
        for (int i = 1 ; i <= n ; ++i) {
            if (i >= a) vis[i] |= vis[i - a];
            if (i >= b) vis[i] |= vis[i - b];
            if (i >= c) vis[i] |= vis[i - c];
            if (vis[i]) ++cnt;
        }
        cout << cnt << '\n';
        for (int i = 1 ; i <= n ; ++i) vis[i] = 0;
    }
    return 0;
}
```

除了动态规划向前找，也可以向后递推 （写代码演示?)，但是要注意不要数组越界。

周赛 9 T3

一道模板动态规划题。难点在于观察并设计状态。

本题状态用的是 `f[i][j]` 表示 堆到第 `i` 个物品高度 `j` 时，增加最多的时间。而我们不超时，即增加的时间 + 原始截止时间 <= 当前时间，即 `f[i][j] + T <= t` 。在满足时间可行的情况下，尝试向后更新

```cpp
#include <iostream>
using namespace std;

const int N = 1003;
const int H = 10001;
int f[N][H];
// 堆到 i 物品 h 高度增加的总时间

int main() {
    int n,T,H;
    cin >> n >> T >> H;

    for (int i = 0 ; i <= n ; ++i)
        for (int j = 1 ; j <= H ; ++j)
            f[i][j] = -1e9;
```

```
    for (int i = 1 ; i <= n ; ++i) {
        int t,h,w;
        cin >> t >> h >> w;
        for (int j = 0 ; j <= H ; ++j) {
            if (f[i - 1][j] >= t - T) {
                // 堆东西，时间不加
                int p = min(j + h, H);
                f[i][p] = max(f[i][p], f[i - 1][j]);
                // 不堆东西，加时间
                f[i][j] = max(f[i][j], f[i - 1][j] + w);
                // cout << f[i][j] << "---" << i << ' ' << j << '\n';
            }
        }
        if (f[i][H] >= t - T) {
            cout << t << '\n';
            return 0;
        }
    }
    cout << -1 << '\n';
    return 0;
}
```

这个做法空间复杂度不是最优，因为需要用到 $O(n \times H)$ 的空间，如果程序限制了我们内存的大小，那么我们就无法解决。解决方案：

- 滚动数组优化。（常用优化，奇数用 1,偶数用 0）
- 压缩掉一个维度。（逆向枚举，更加极端，但是适用条件更为苛刻）
- 必须是只依赖上一次的数据才行。（后面背包问题还会再遇到一次）。

背包问题

背包问题是**一类**经典的动态规划问题。

最简单的问题：01 背包问题

- 给定 n 个物品，每个物品有重量 w_i 和价值 v_i 。我们限制选取的物品重量之和不超过 W （即背包只装得下一定的物品）。在这样的情况下，请求出我们能获得的最大价值之和。

How to solve?（这一段估计要慢慢讲）

- 二维数组动态规划 $f_{i,j}$ 表示，选了 i 个物品，重量不超过 j 时，能获得的最大价值。
- 先枚举每个物品(外层循环)，再枚举每个重量。每个重量有两种可能的选择
    1. 不选这个物品，那么最大价值就和 $i-1$ 的时候一样 $f_{i,j} = f_{i-1,j}$
    1. 选这个物品，那么最大价值比 $i-1, j-w$ 时的最大值，还有额外多这个物品的价值
  $f_{i,j} = f_{i-1,j-w} + v$
- 对于这两个选择，取 max 即可。

熟能生巧：开始解题 :)

采药(1)

模板题，让他们自己写一写。

- 演示一遍简单 std
- 再演示一遍前面说的滚动数组（因为只依赖前一项，全部加个 & 1 即可，奇数用 1 ， 偶数用 0）
- 可以可以尝试压缩掉这个数组。
  - 注意到只依赖前一项比自己小的项，因此
  - 不可以直接压缩，必须倒序枚举保证正确性。
  - 记住这个写法，这是 01 背包最常用、最高效的写法！
  - excel 画图演示正确性即可。

```
// 朴素版本
#include <iostream>
using namespace std;

const int N = 1003;
int f[N][N];

int main() {
    int T,n;
    cin >> T >> n;
    for (int i = 1 ; i <= n ; ++i) {
        /* 注意，是 0 开始 */
        int t, w;
        cin >> t >> w;
        for (int j = 0 ; j <= T ; ++j) {
            if (j >= t)
                f[i][j] = max(f[i - 1][j], f[i - 1][j - t] + w);
            else
                f[i][j] = f[i - 1][j];
        }
    }
    cout << f[n][T] << '\n';
    return 0;
}
```

```
// 常用版本
#include <iostream>
using namespace std;

const int N = 1003;
int f[N];

int main() {
    int T,n;
    cin >> T >> n;
    for (int i = 1 ; i <= n ; ++i) {
        int t, w;
        cin >> t >> w;
        for (int j = T ; j >= t ; --j) {
```

```
                f[j] = max(f[j], f[j - t] + w);
            }
        }
        cout << f[T] << '\n';
        return 0;
    }
```

最大约数和(7)

提示：背包问题。（思考怎么转化为已知问题）（转化完就是套模板的事情了）

(直接在 T1 模板上修改即可)

```
#include <iostream>
using namespace std;

const int N = 1003;
int f[N];

// 约数个数
int count(int n) {
    int ans = 0;
    for (int i = 1 ; i < n ; ++i)
        if (n % i == 0) ans += i;
    return ans;
}

int main() {
    int n;
    cin >> n;
    for (int i = 1 ; i <= n ; ++i) {
        int t = i;          //  开销
        int w = count(i);   //  约数之和

        for (int j = n ; j >= t ; --j) {
            f[j] = max(f[j], f[j - t] + w);
        }
    }
    cout << f[n] << '\n';
    return 0;
}
```

超市 2.0 (4)

(直接做 2.0 ，1.0 就不做了。)

简单来说，有两个维度的约束。在这样的情况下，需要转换模型。幸运的是，转移方程还是大差不差。我们照样可以先枚举每个物品，再枚举约束，完成问题。

空间复杂度优化:(?)

- 倒序循环法由于现在是两个维度的约束，简单的倒序不一定正确（但可以证明安全）

- 滚动数组法依然可用，非常的安全，非常的保险（不用担心正确性）。

```
#include <iostream>
using namespace std;

const int N = 103;
int w[N], v[N], h[N];
int f[N][N];

signed main() {
    int n, W, m ;
    cin >> n >> W >> m;
    for (int i = 1; i <= n; ++i) cin >> w[i] >> v[i] >> h[i];
    for (int i = 1; i <= n; ++i) {
        for (int j = W; j >= w[i]; --j) {
            for (int k = m ; k >= h[i] ; --k) {
                f[j][k] = max(f[j][k], f[j - w[i]][k - h[i]] + v[i]);
            }
        }
    }
    cout << f[W][m] << endl;
    return 0;
}
```

拓展阅读

背包 9 讲。有更多进阶的背包问题，有助于理解动态规划。

- 完全背包
- 二进制压缩