# 递推

递推法常常用来解决

- 计数问题
- 组合最优化问题

dynamic programming，DP

# 动 态 规 划

一种解题技术或者说算法设计方法。

- 狭义：用递推法解决组合最优化问题。
- 广义：和递推法是同义词。

# 递推三要素

- 子问题（状态）
- 递推式（状态转移）
- 边界条件（初始/边界状态）

# 例一

有 $N$ 个东西排成一行，要从中选一些，可以一个也不选。相邻的两个东西不能都选。有多少种方法。

# 解法：递推

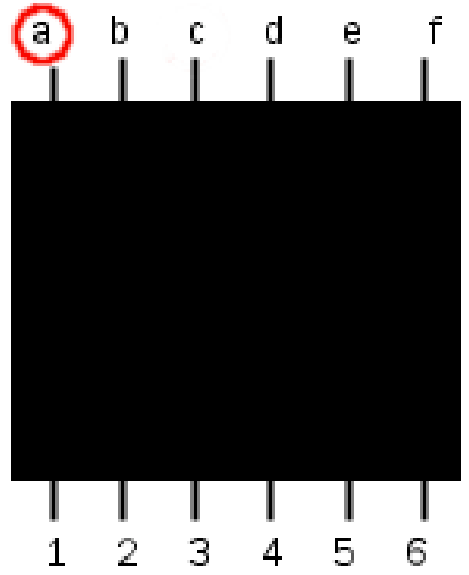
子问题：从前 $i$ 个东西里选一些，相邻两个不能都选，有多少种方法？
这里，$i = 0, 1, 2, \ldots, N$。把答案记作 $f[i]$。

目标：$f[N]$。

边界条件：$f[0] = 1$，$f[1] = 2$。

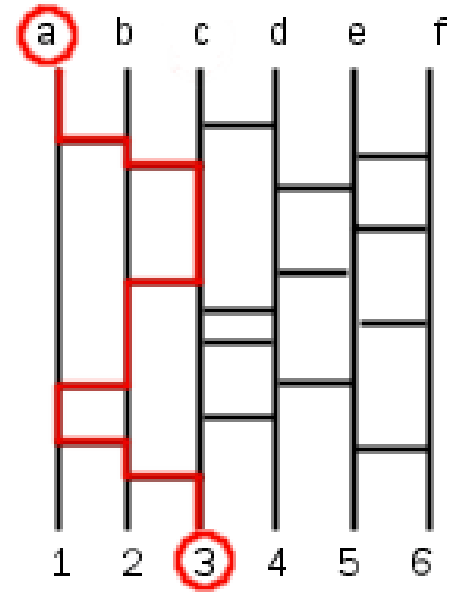
递推式：$f[i] = f[i-1] + f[i-2]$，$i \geq 2$。

# 阿弥陀签




又称鬼脚图、爬梯游戏，是一种游戏，也是一种简易决策方法，常被拿作抽签或决定分配组合。

先画几条平行的竖线，再画一些横线，每条横线连接相邻两条竖线。两条横线不能相交。




玩法：

1. 把画的横线盖住，选一条竖线。
2. 揭开盖子，从所选竖线的上端往下走，遇到横线就走过去，直到走到某条竖线的下端。

# 例二 abc113_d Number of Amidakuji

阿 弥 陀 签

有 $W$ 条长度是 $H+1$ 厘米的竖线，要画一些横线做成一个阿弥陀签。每条横线的两端点到竖线上端的距离必须是 $1$，$2$，……，或 $H$ 厘米。有多少种画法满足：

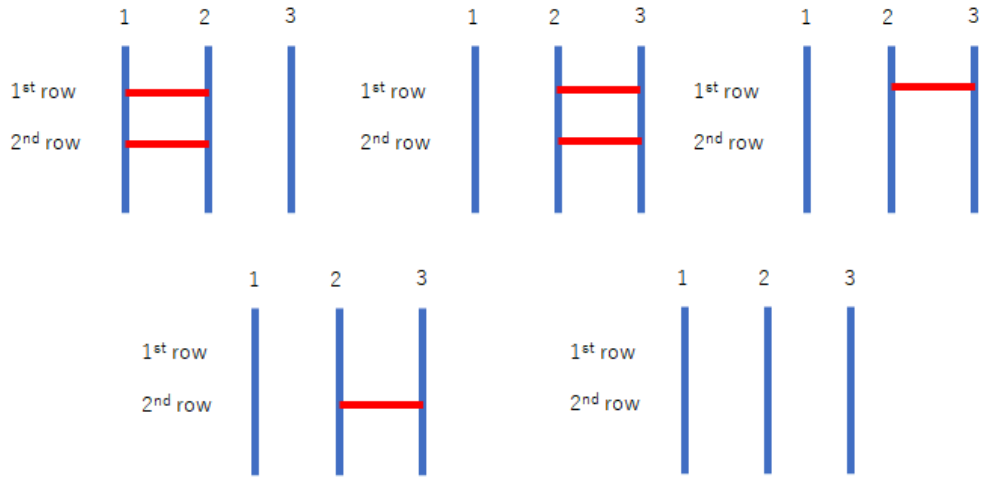
- 从第一条竖线的上端出发最后会走到第 $K$ 条竖线的下端。

输出答案模 $10^9+7$。

## 限制

- $1 \leq H \leq 100$
- $1 \leq W \leq 8$
- $1 \leq K \leq W$

## 样例




The number of Amidakuji – For (H, W, K) = (2, 3, 1)

# 位置，一步

用一对整数 $(i, j)$ 表示位置：在第 $j$ 条竖线上，距离竖线的上端 $i$ 厘米。
$0 \leq i \leq H$，$1 \leq j \leq W$。

初始位置 $(0, 1)$，目标位置 $(H, K)$。

设当前位置是 $(i, j)$，称下述动作为**一步**

- 向下移动 1 厘米，到 $(i+1, j)$。若 $(i+1, j)$ 处有横线，就沿着横线走过去。

# 递推

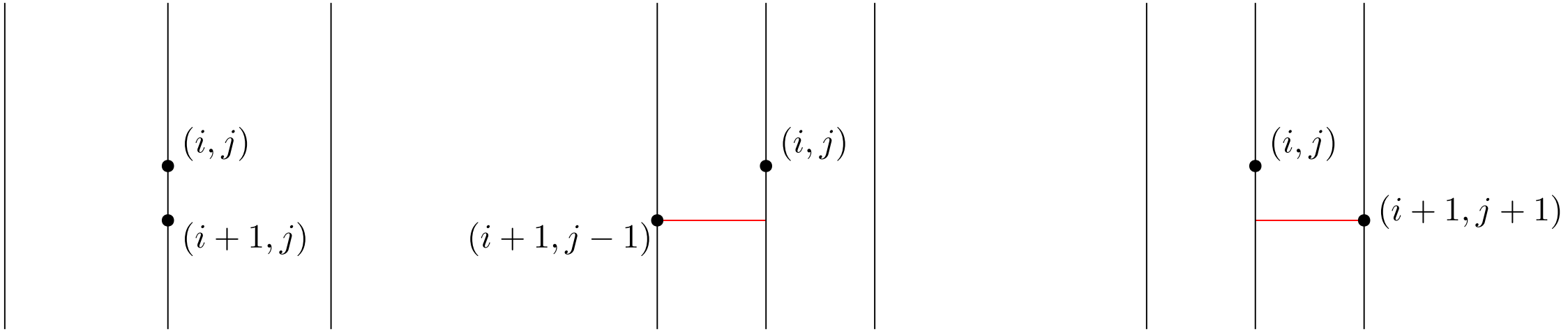
各行相互**独立**。从上到下考虑每一行。

定义 $g[i][j]$：对前 $i$ 行，有多少种画法使得

- 从 $(0,1)$ 经过 $i$ **步**到达 $(i,j)$。

边界条件：$g[0][1]=1$。

转移：

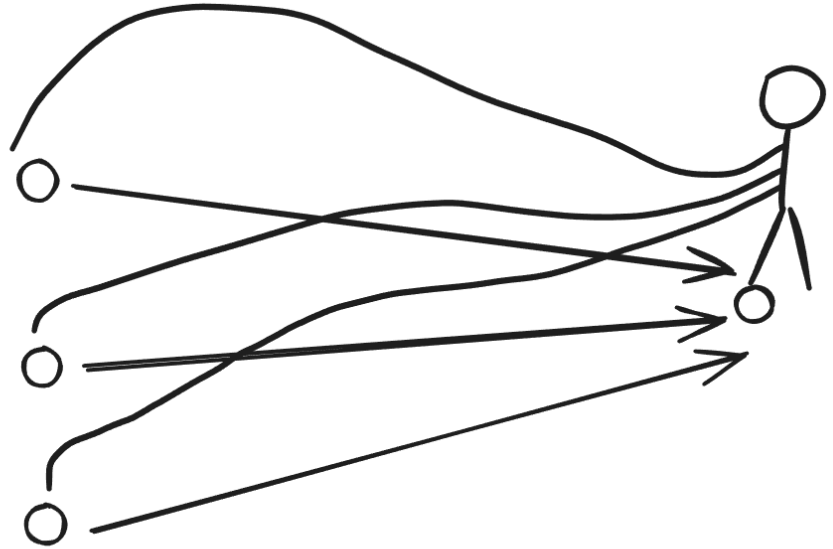


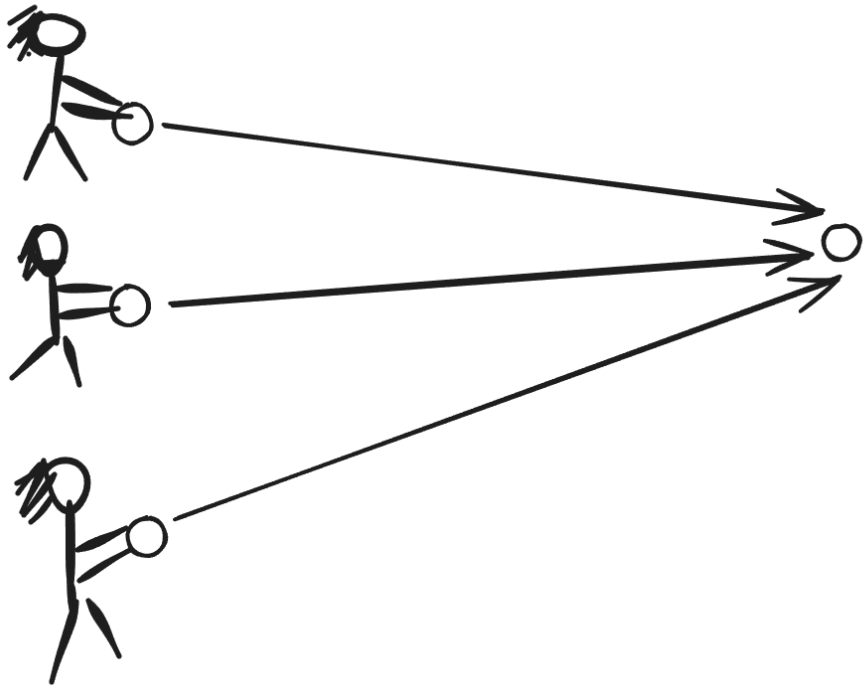
# 代码

```
const int mod = 1e9 + 7;
long long g[101][8], F[10];
long long f(int i) { return i < 0 ? 1 : F[i]; }
void solve() {
  int h, w, k; cin >> h >> w >> k; k--;// 竖线从0开始编号。
  F[0] = 1, F[1] = 2;
  for (int i = 2; i < 10; i++) F[i] = F[i - 1] + F[i - 2];
  g[0][0] = 1;
  for (int i = 0; i < h; i++)
    for (int j = 0; j < w; j++) {
      g[i][j] %= mod;
      g[i + 1][j] += g[i][j] * f(j - 1) * f(w - j - 2);
      if (j > 0)
        g[i + 1][j - 1] += g[i][j] * f(j - 2) * f(w - j - 2);
      if (j + 1 < w)
        g[i + 1][j + 1] += g[i][j] * f(j - 1) * f(w - j - 3);
    }
  cout << g[h][k] % mod << '\n';
}
```

# DP 的两种实现方式

- 前向DP（刷表法，push DP）
- 后向DP（填表法，pull DP）





# 序列上的动态规划

# 背包类型动态规划

- 前置知识：01背包问题。

# dp_e Knapsack 2

有 $N$ 个物品。编号 $1, 2, \ldots, N$。第 $i$ 个物品的重量是 $w_i$，价值是 $v_i$。

太郎要从 $N$ 个物品中选一些放进背包带回家。背包的容量是 $W$，这意味着所选物品的重量之和不能超过 $W$。

求所选物品的价值之和的最大值。

### 限制

- $1 \leq N \leq 100$
- $1 \leq W \leq 10^9$
- $1 \leq w_i \leq W$
- $1 \leq v_i \leq 10^3$
- 以上值都是整数。

# 子问题

- 从前 $i$ 个物品中选一些，所选物品的总重量不超过 $j$，所选物品的总价值的最大值。

这里，$0 \leq i \leq N$，$0 \leq j \leq W$，子问题有 $(N+1)(W+1)$ 个，太多了。

注意到 $v_i \le 10^3$，$N \le 100$，这意味着全部物品的价值之和不超过 $10^5$。

## 新的子问题

- 从前 $i$ 个物品中选一些，所选物品的总价值等于 $j$，所选物品的**总重量的最小值**。

这里，$0 \le i \le N$，$0 \le j \le 10^5$。

把这个问题的答案记作 $f[i][j]$，若无法做到，令 $f[i][j] = W + 1$。

边界条件：$f[0][0] = 0$，$f[0][j] = W + 1$，$j = 1, \ldots, 10^5$。

递推式：对 $i = 1, \ldots, N$，有

$$f[i][j] = \begin{cases} f[i-1][j], & 0 \le j < v_i, \\ \min(f[i-1][j], f[i-1][j-v_i] + w_i), & v_i \le j \le 10^5. \end{cases}$$

# 代码

```
void solve() {
  int N, W; cin >> N >> W;
  const int S = 1e5;
  vector<int> f(S + 1, W + 1);
  f[0] = 0;
  for (int i = 1; i <= n; i++) {
    int w, v;
    cin >> w >> v;
    for (int j = S; j >= v; j--)
      f[j] = min(f[j], f[j - v] + w);
  }
  for (int j = S; j >= 0; j--)
    if (f[j] <= W) {
      cout << j << "\n";
      break;
    }
}
```

# abc375_e 3 Team Division

有 $N$ 个人分成了三队。人编号 $1, 2, \ldots, N$，队编号 $1, 2, 3$。现在，人 $i$ 属于队 $A_i$。

每人有一个强度值，人 $i$ 的强度是 $B_i$。一个队的强度定义为其成员的强度之和。判断能否通过改变某些人所属的队使得三个队的强度相同。若可能，输出最少要几个人换队。否则输出 $-1$。

### 限制

- $3 \leq N \leq 100$
- $A_i \in \{1, 2, 3\}$
- $1 \leq B_i$
- $\sum_{i=1}^{N} B_i \leq 1500$
- 上述值都是整数。

# 分析

必要条件 1：$3$ 整除 $\sum_{i=1}^{N} B_i$。

令 $M = (\sum_{i=1}^{N} B_i)/3$。注意到 $M \leq 500$。

必要条件 2：$\max_{i=1}^{N} B_i \leq M$。

# 子问题

把前 $i$ 个人分到三个队，1队的强度是 $j$，2队的强度是 $k$，最少有几个人换队。

把它的答案记作 $f[i][j][k]$。若无法做到，令 $f[i][j][k] = N + 1$。
这里 $0 \le i \le N$，$0 \le j, k \le M$。

目标：$f[N][M][M]$。

边界条件：$f[0][0][0] = 0$，$f[0][i][j] = N + 1$，$i \neq 0$ 或 $j \neq 0$。

递推式：

$$f[i][j][k] = \min \begin{cases} f[i-1][j-B_i][k] + [A_i \neq 1], & B_i \le j \\ f[i-1][j][k-B_i] + [A_i \neq 2], & B_i \le k \\ f[i-1][j][k] + [A_i \neq 3] \end{cases}$$

# 代码

```cpp
template <typename T, typename U>
bool chmin(T& a, U b) {
  if (b < a) { a = b; return true; }
  return false;
  // return b < a ? a = b, true : false;
}

void solve() {
  int n; cin >> n;
  vector<int> a(n), b(n);
  for (int i = 0; i < n; i++) cin >> a[i] >> b[i];
  int m = accumulate(b.begin(), b.end(), 0);
  if (m % 3) { cout << -1 << '\n'; return; }
  m /= 3;
  if (*max_element(b.begin(), b.end()) > m) { cout << -1 << '\n'; return; }
  vector<vector<int>> dp(m + 1, vector<int>(m + 1, n + 1));
  dp[0][0] = 0;
  for (int i = 0; i < n; i++) {
    for (int j = m; j >= 0; j--)
      for (int k = m; k >= 0; k--) {
        dp[j][k] += (a[i] != 3);
        if (j >= b[i])
          chmin(dp[j][k], dp[j - b[i]][k] + (a[i] != 1));
        if (k >= b[i])
          chmin(dp[j][k], dp[j][k - b[i]] + (a[i] != 2));
      }
  }
  cout << (dp[m][m] > n ? -1 : dp[m][m]) << '\n';
}
```

# abc383_f Diversity

商店里有 $N$ 个商品在售，第 $i$ 个商品的价格是 $P_i$ 元，效用是 $U_i$，颜色是 $C_i$。

你要从这些商品中买一些（可以一个都不买）。所买物品的价格之和不得超过 $X$ 元。

你的满意度是 $S + T \times K$，其中 $S$ 是所买物品的效用之和，$T$ 是所买物品的不同颜色的数量，$K$ 是给定的常数。

求最大满意度。

限制

- $1 \leq N \leq 500$
- $1 \leq X \leq 50000$
- $1 \leq K \leq 10^9$
- $1 \leq P_i \leq X$
- $1 \leq U_i \leq 10^9$
- $1 \leq C_i \leq N$
- 上述值都是整数。

# 提示

把物品按颜色分组。一次处理一批同色的物品。

# 子问题

从前 $i$ 种颜色的物品中选一些，所选物品的总价格不超过 $j$，满意度的最大值。

把这个量记作 $f[i][j]$。这里，$0 \le i \le N$，$0 \le j \le X$。

目标：$f[N][X]$。

边界条件：$f[0][j] = 0$，$0 \le j \le X$。

递推式：

对每种颜色 $i = 1, \ldots, N$，

1. 置 $f[i][j] = f[i-1][j]$，$0 \le j \le X$。
2. 对每个颜色是 $i$ 的物品 $(P, U)$，按 $j = X, X-1, \ldots, P$ 的顺序，置

$$f[i][j] = \max(f[i][j], f[i-1][j-P] + U + K, f[i][j-P] + U)$$

# 代码

```
void solve() {
  int n, x, k; cin >> n >> x >> k;
  vector<vector<pair<int,int>>> group(n + 1);
  for (int i = 0; i < n; i++) {
    int p, u, c; cin >> p >> u >> c;
    group[c].push_back({p, u});
  }
  vector<long long> f(x + 1, 0), g = f;
  for (int c = 1; c <= n; c++) {
    for (auto [p, u]: group[c]) //structured binding，需要C++17
      for (int i = x; i >= p; i--)
        chmax(g[i], max(g[i - p] + u, f[i - p] + u + k));
    f = g;
  }
  cout << f[x] << '\n';
}
```

# CSES1665 Coding Company

给你 $N$ 个整数 $T_1, \ldots, T_N$。要把它们分成若干组，每一组的极差之和不超过 $X$。求分组方法数，模 $10^9 + 7$。

一组整数的极差是最大值减最小值。

### 限制

- $1 \leq N \leq 100$
- $0 \leq X \leq 5000$
- $0 \leq T_i \leq 100$

### 样例

```
3 2
2 5 3
```

```
3
```

分组方法：

- (2), (3), (5)
- (2, 3), (5)
- (2), (3, 5)

把 $N$ 个数从大到小排序。

子问题

$f[i][j][k]$：对前 $i$ 个数分组，尚未分好的组（最小值尚未出现）有 $j$ 个，且满足

- 已分好的组的极差之和 + 未分好的组的最大值之和 $\leqslant k$

有多少种方法?

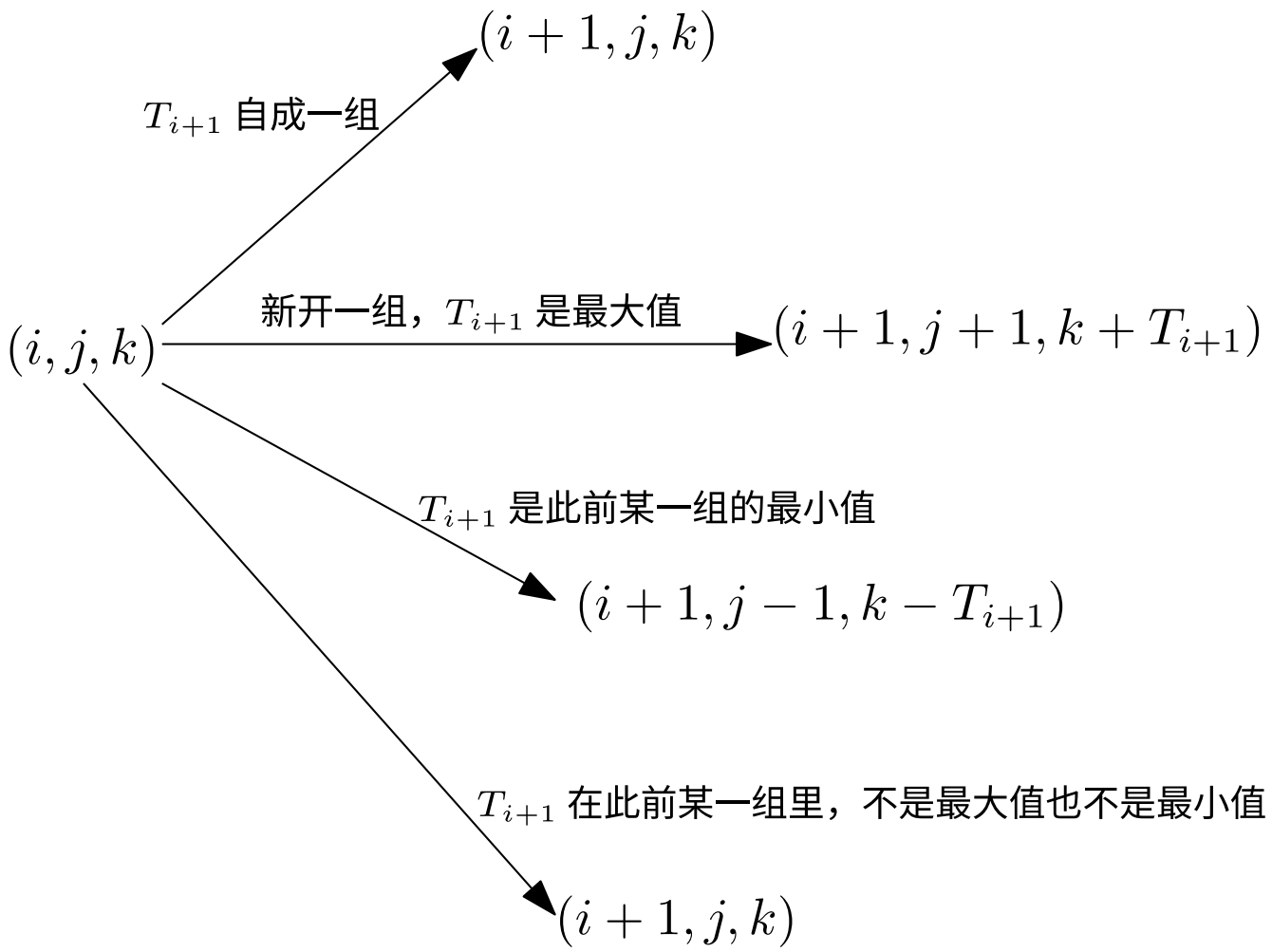
注意到

- $0 \leq i \leq N$
- $0 \leq j \leq \lfloor N/2 \rfloor \leq 50$
- $0 \leq k \leq X + \lfloor N/2 \rfloor \times \max(T_1, \ldots, T_N) \leq 10000$

目标：$f[N][0][X]$。

边界条件：$f[0][0][k] = 1$ $(0 \leq k \leq X)$，$f[0][j][\cdot] = 0$ $(j \neq 0)$。

# 状态转移


$(i+1,j,k)$
$T_{i+1}$ 自成一组
新开一组，$T_{i+1}$ 是最大值
$(i,j,k)$
$(i+1,j+1,k+T_{i+1})$
$T_{i+1}$ 是此前某一组的最小值
$(i+1,j-1,k-T_{i+1})$
$T_{i+1}$ 在此前某一组里，不是最大值也不是最小值
$(i+1,j,k)$

# 代码

```
const int mod = 1e9 + 7;
long long dp[101][51][10001];

void solve() {
  int n, x; cin >> n >> x;
  for (int k = 0; k <= x; k++) dp[0][0][k] = 1;
  vector<int> t(n);
  for (int i = 0; i < n; i++) cin >> t[i];
  sort(t.rbegin(), t.rend());
  for (int i = 0; i < n; i++) {
    for (int j = 0; j <= min(i, n - i); j++) {
      for (int k = 0; k <= 10000; k++) {
        if (dp[i][j][k] == 0) continue;
        dp[i][j][k] %= mod;
        // t[i] 自成一组
        dp[i + 1][j][k] += dp[i][j][k];
        // 新开一组，t[i]是其中的最大值
        if (j + 1 <= 50)
          dp[i + 1][j + 1][k + t[i]] += dp[i][j][k];
        // t[i] 是某一组的最小值
        if (j)
          dp[i + 1][j - 1][k - t[i]] += j * dp[i][j][k];
        // t[i] 既不是某一组的最大值也不是某一组的最小值
        if (j)
          dp[i + 1][j][k] += j * dp[i][j][k];
      }
    }
  }
  cout << dp[n][0][x] % mod << '\n';
}
```

# abc288_e Wish List

商店里有 $N$ 个商品，编号为商品 1，商品 2，…，商品 $N$。商品 $i$ 的价格是 $A_i$ 元。
高桥想要 $M$ 个商品：商品 $X_1$，商品 $X_2$，…，商品 $X_M$。

他重复下述操作直到买齐他想要的商品。

> 令 $r$ 为当前尚未卖出去的商品的数量。选择一个整数 $j$ 满足 $1 \leq j \leq r$，买剩余物品中编号第 $j$ 小的那个物品，花的钱是那个物品的价格再加上 $C_j$ 元。

求高桥最少要花多少钱。

高桥也可以买他不想要的物品。

### 限制

- $1 \leq M \leq N \leq 5000$
- $1 \leq A_i, C_i \leq 10^9$
- $1 \leq X_1 < X_2 < \cdots < X_M \leq N$

# 关键性质

- 编号大于 $X_M$ 的商品不用考虑。不妨置 $N = X_M$。
- 何时买商品 $i$ 不影响对前 $i-1$ 个商品中任何一个的购买。
- 设最终在前 $i-1$ 个商品中买了 $j$ 个，如果也买了商品 $i$，那么买它时，它的编号排名可以是 $i-j, i-j+1, \ldots, i$ 中的任何一个。所以买商品 $i$ 的花费是 $A_i + \min_{i-j \le k \le i} C_k$。

# 子问题

$f[i][j]$：在前 $i$ 个商品中买 $j$ 个，想要的商品的都买了，最少要花多少钱?

这里，$0 \le i \le N$，$0 \le j \le N$。
状态 $(i, j)$ 要满足 $(\text{前}i\text{个物品中必买的物品的数量}) \le j \le i$ 才合理。
若状态 $(i, j)$ 不合理，令 $f[i][j] = \infty$。

目标：$\max_{M \le j \le N} f[N][j]$。

边界条件：$f[i][0] = 0$ $(0 \le i < X_1)$

递推式：对 $1 \le i \le N$，$1 \le j \le N$

$$f[i][j] = \begin{cases} f[i-1][j-1] + A_i + \min_{i-(j-1) \le k \le i} C_k, & \text{商品}i\text{必买}, \\ \min(f[i-1][j], f[i-1][j-1] + A_i + \min_{i-(j-1) \le k \le i} C_k), & \text{商品}i\text{不必买}. \end{cases}$$

# 代码

```
void solve() {
  int n, m; cin >> n >> m;
  vector<int> a(n + 1), c(n + 1), x(m);
  for (int i = 1; i <= n; i++) cin >> a[i];
  for (int i = 1; i <= n; i++) cin >> c[i];
  for (int i = 0; i < m; i++) cin >> x[i];
  n = x.back();
  vector<bool> must_buy(n + 1);
  for (int i : x) must_buy[i] = true;

  const long long INF = LLONG_MAX / 2;
  vector<vector<long long>> dp(n + 1, vector<long long>(n + 1, INF));
  for (int i = 0; i < x[0]; i++) dp[i][0] = 0;
  int must_cnt = 0;

  for (int i = 1; i <= n; i++) {
    must_cnt += must_buy[i];
    int min_c = INT_MAX;
    for (int j = 1; j <= i; j++) {
      chmin(min_c, c[i - (j - 1)]);
      if (j < must_cnt) continue;
      if (must_buy[i])
        dp[i][j] = dp[i - 1][j - 1] + a[i] + min_c;
      else
        dp[i][j] = min(dp[i - 1][j], dp[i - 1][j - 1] + a[i] + min_c);
    }
  }
  long long ans = INF;
  for (int i = m; i <= n; i++) chmin(ans, dp[n][i]);
  cout << ans << '\n';
}
```

# 网格上的动态规划

# abc210_d National Railway

高桥王国可表示为一个 $H \times W$ 的网格。第 $i$ 行第 $j$ 列的格子记作 $(i, j)$。

高桥要建一条铁路。建铁路分为两个阶段：

- 首先，选两个**不同的**格子，在每个上面建一座车站。在格子 $(i, j)$ 上建车站的花费是 $A_{i,j}$ 元。
- 然后，建设连接两座车站的轨道。建设连接格子 $(i, j)$ 和 $(i', j')$ 的轨道要花费 $C \times (|i - i'| + |j - j'|)$ 元。

求建设铁路的总花费的最小值。

### 限制

- $2 \leq H, W \leq 1000$
- $1 \leq C, A_{i,j} \leq 10^9$

# 分析

设两座车站建在格子 $(i,j)$ 和 $(i',j')$。不失一般性，假设 $i' \le i$。
分两种情况：

- $j' \le j$
- $j' > j$

当 $j' \le j$ 时，总花费 $A_{i,j} + A_{i',j'} + C \times (|i-i'| + |j-j'|)$ 可写成

$$(A_{i,j} + C(i+j)) + (A_{i',j'} - C(i'+j'))$$

固定 $(i,j)$，问题化为求 $\min_{1 \le i' \le i, 1 \le j' \le j, (i',j') \ne (i,j)} A_{i',j'} - C(i'+j')$。

# 子问题

$f[x][y]$：$\min_{1\le i\le x,1\le j\le y} A_{i,j} - C(i+j)$

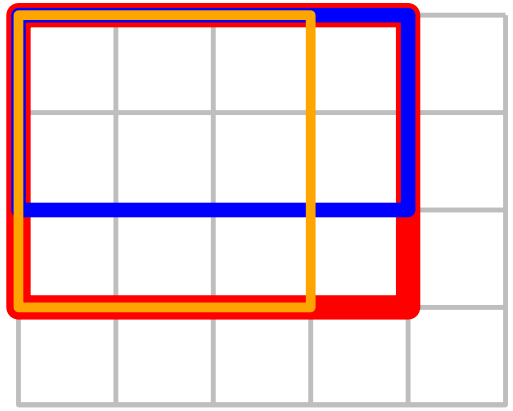
递推式：

$$f[x][y] = \min(f[x-1][y], f[x][y-1], A_{x,y} - C(x+y))$$

边界条件：

$$f[0][y] = \infty \ (0 \le y \le W)$$
$$f[x][0] = \infty \ (0 \le x \le H)$$

对 $j' > j$ 情形的处理是类似的。



# 代码

```
int A[1005][1005];
long long f[1005][1005];
void solve() {
  int H, W, C; cin >> H >> W >> C;
  for (int i = 1; i <= H; i++)
    for (int j = 1; j <= W; j++)
      cin >> A[i][j];
  memset(f, 0x3f, sizeof f);
  long long ans = LLONG_MAX;
  for (int i = 1; i <= H; i++)
    for (int j = 1; j <= W; j++) {
      chmin(ans, min(f[i - 1][j], f[i][j - 1]) + A[i][j] + (i + j) * 1LL * C);
      f[i][j] = min({f[i -1][j], f[i][j - 1], A[i][j] - (i + j) * 1LL * C});
    }
  for (int i = 1; i <= H; i++)
    for (int j = W; j >= 1; j--) {
      chmin(ans, f[i][j + 1] + A[i][j] + (i - j) * 1LL * C);
      f[i][j] = min(f[i - 1][j], f[i][j + 1], A[i][j] + (j - i) * 1LL * C);
    }
  cout << ans << '\n';
}
```

# 节省空间的写法

```
void solve() {
  int H, W, C; cin >> H >> W >> C;
  vector<int> A(W + 1);
  const long long INF = LLONG_MAX / 2;
  vector<long long> l(W + 1, INF), r(W + 2, INF);
  long long ans = LLONG_MAX;
  for (int i = 1; i <= H; i++) {
    for (int j = 1; j <= W; j++)
      cin >> A[j];
    for (int j = 1; j <= W; j++) {// 从左向右
      chmin(ans, min(l[j - 1], l[j]) + A[j] + (i + j) * 1LL * C);
      l[j] = min({l[j - 1], l[j], A[j] - (i + j) * 1LL * C});
    }
    for (int j = W; j >= 1; j--) {// 从右向左
      chmin(ans, r[j + 1] + A[j] + (i - j) * 1LL * C);
      r[j] = min({r[j + 1], r[j], A[j] + (j - i) * 1LL * C});
    }
  }
  cout << ans << '\n';
}
```

# abc358_g AtCoder Tour

有一个 $H \times W$ 的网格。把第 $i$ 行第 $j$ 列的格子记作 $(i, j)$。

一开始，高桥在格子 $(S_i, S_j)$ 里，他将重复下述动作 $K$ 次：

- 他或者待在当前格子里或者移动到一个相邻的格子里。这个动作过后，若他在格子 $(i, j)$ 里，他将获得 $A_{i,j}$ 点快乐。

求高桥可能获得的总快乐点数的最大值。

这里，格子 $(x', y')$ 和格子 $(x, y)$ 相邻当且仅当 $|x - x'| + |y - y'| = 1$。

### 限制

- $1 \leq H, W \leq 50$
- $1 \leq K, A_{i,j} \leq 10^9$
- $1 \leq S_i \leq H$
- $1 \leq S_j \leq W$

# 分析

为了使总快乐点数尽可能大，高桥可采取下述策略：

1. 选一个格子作终点，到终点后就待在那儿不动。
   终点的点数大于所有途经格子的点数。
2. 沿着某条**简单路径**走到终点，中途不停留。

注意：不必用最少的步数走到终点。例如 $A = \begin{pmatrix} 1 & 4 \\ 1 & 4 \\ 5 & 4 \end{pmatrix}$，起点是 $(1, 1)$，$K = 10$。

注意到一共有 $HW$ 个格子，走到终点所需的步数至多 $HW - 1$。我们定义以下子问题

- 从起点走 $t$ 步到达格子 $(i, j)$ 能获得的总点数的最大值。

这里，$0 \leq t \leq \min(K, HW - 1)$。

# 代码

```cpp
int dir[4][2] = {1, 0, -1, 0, 0, 1, 0, -1};
int a[50][50];
long long f[50][50], g[50][50];
void solve() {
  int h, w, k, si, sj;
  cin >> h >> w >> k >> si >> sj;
  si--, sj--;
  for (int i = 0; i < h; i++)
    for (int j = 0; j < w; j++)
      cin >> a[i][j];

  memset(f, -1, sizeof f);
  f[si][sj] = 0;

  long long ans = (long long) k * a[si][sj];

  for (int t = 1; t <= min(k, h * w - 1); t++) {
    memset(g, -1, sizeof g);
    for (int i = 0; i < h; i++)
      for (int j = 0; j < w; j++)
        if (f[i][j] >= 0)
          for (auto [x, y] : dir) { //structured binding，需要C++17
            int ni = i + x, nj = j + y;
            if (ni >= 0 && ni < h && nj >= 0 && nj < w)
              chmax(g[ni][nj], f[i][j] + a[ni][nj]);
          }

    for (int i = 0; i < h; i++)
      for (int j = 0; j < w; j++)
        if (g[i][j] != -1)
          chmax(ans, g[i][j] + (long long) a[i][j] * (k - t));
    swap(f, g);
  }
  cout << ans << '\n';
}
```

# P7074 方格取数

有一个 $N \times M$ 的网格，第 $i$ 行第 $j$ 列的格子里有一个整数 $A_{i,j}$。

有一只小熊想从图的左上角走到右下角。每一步只能向上、向下或向右走一格，并且不能重复经过已经走过的方格，也不能出界。

小熊会取走经过的方格中的整数，求它能取到的整数之和的最大值。

### 限制

- $1 \leq N, M \leq 10^3$
- $-10^4 \leq A_{i,j} \leq 10^4$

# 动态规划

- $f[i][j]$：从 $(1,1)$ 到 $(i,j)$ 的路径上的数字之和的最大值。
- 答案是 $f[N][M]$。
- 递推式?

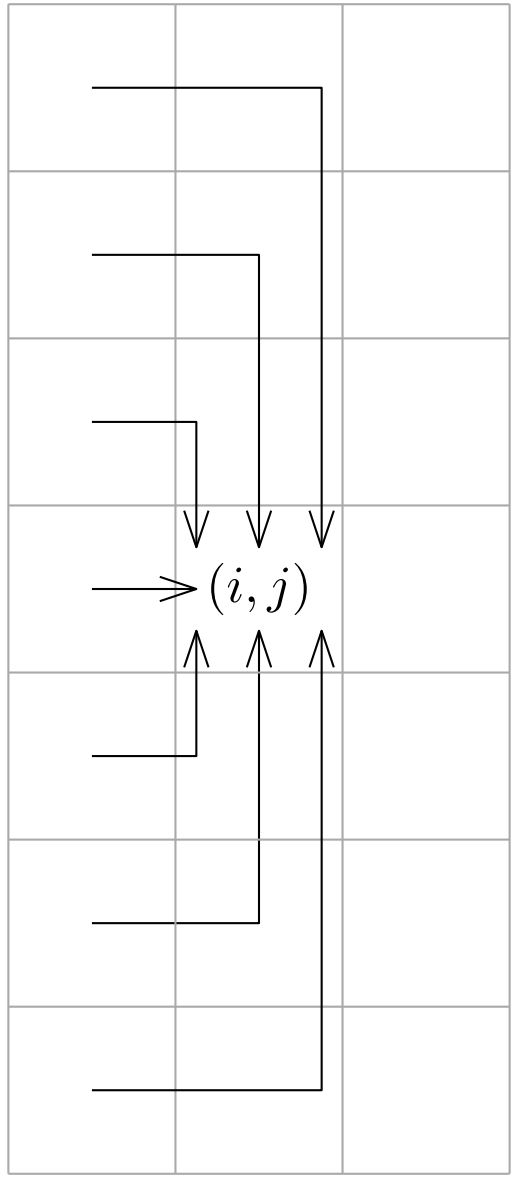
# 错误的递推式

$$f[i][j] = A[i][j] + \max(f[i][j-1], f[i-1][j], f[i+1][j])$$

从 $(1,1)$ 到 $(i-1,j)$ 或 $(i+1,j)$ 的最优路径可能经过 $(i,j)$。

# 状态转移

从第 $j-1$ 行的某个格子 $(i', j-1)$ 先向右一步走到 $(i', j)$ 再向上或向下走到 $(i, j)$。



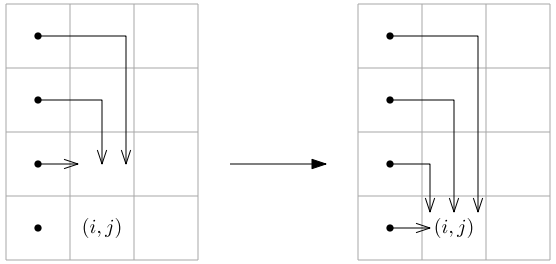
# 分两步计算

把从 $(1,1)$ 到 $(i,j)$ 的路径分成两类：

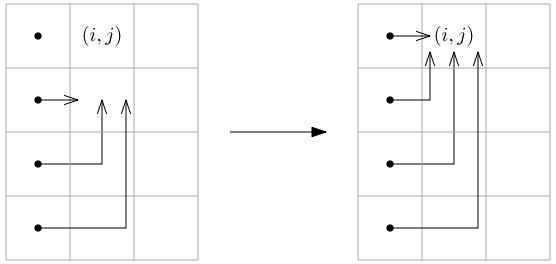
- 到 $(i,j)$ 之后可以继续向下走
- 到 $(i,j)$ 之后可以继续向上走

分两步计算一列格子的 DP 值：

1. 从上到下




2. 从下到上



# 代码

```cpp
const int maxn = 1005;
int a[maxn][maxn];
long long f[maxn][maxn];

int main() {
  int n, m;
  cin >> n >> m;
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= m; j++)
      cin >> a[i][j];
  for (int r = 1; r <= n; r++) //边界条件：第一列
    f[r][1] = a[r][1] + f[r - 1][1];
  for (int c = 2; c <= m; c++) {
    long long best = LLONG_MIN;
    for (int r = 1; r <= n; r++) {
      best = max(best, f[r][c - 1]) + a[r][c];
      f[r][c] = best;
    }
    best = LLONG_MIN;
    for (int r = n; r >= 1; r--) {
      best = max(best, f[r][c - 1]) + a[r][c];
      f[r][c] = max(f[r][c], best);
    }
  }
  cout << f[n][m] << "\n";
  return 0;
}
```

# LeetCode 741 Cherry Pickup

有一个 $N \times N$ 的网格。行、列都从 $0$ 到 $N-1$ 编号。
把第 $i$ 行第 $j$ 列的格子记作 $(i, j)$。每个格子里是一个樱桃，空白，或荆棘，分别用整数 $1$，$0$，$-1$ 表示。格子 $(i, j)$ 的状态以整数 $A_{i,j}$ 表示。

求经过下述过程最多能收集多少个樱桃：

- 从 $(0, 0)$ 出发，每一步向下或向右，走到 $(N-1, N-1)$，再从 $(N-1, N-1)$ 出发，每一步向上或向左，走到 $(0, 0)$；不能走进有荆棘的格子。若经过一个有樱桃的格子，就收集一颗樱桃，然后这个格子变成空白。

### 限制

- $1 \leq N \leq 50$
- $A_{1,1} \neq -1$
- $A_{N,N} \neq -1$

# 思路

题目描述的过程相当于

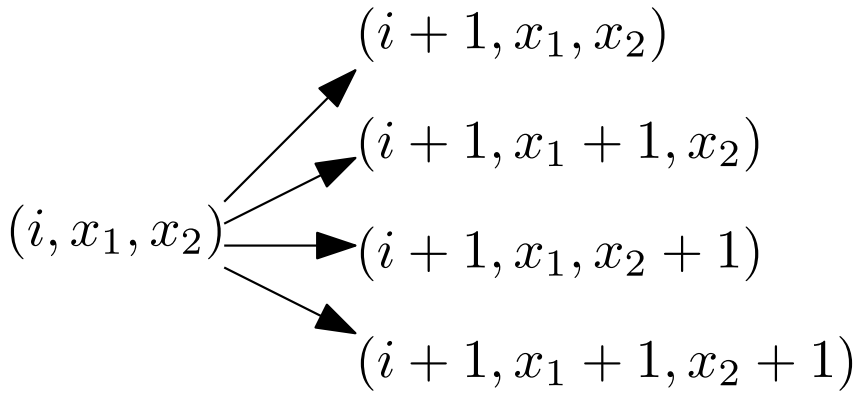
- 甲乙两人同时从 $(0,0)$ 出发，每一步向右或向下，走到 $(N-1,N-1)$。

子问题

$f[i][x_1][x_2]$：$i$ 步之后甲走到 $(x_1,k-x_1)$，乙走到 $(x_2,k-x_2)$，最多能收集多少个樱桃?

边界条件：$f[0][0][0]=A_{0,0}$

状态转移：



# 代码

```
class Solution {
 public:
  int cherryPickup(vector<vector<int>> &grid) {
    int n = grid.size();
    vector<vector<int>> dp(n, vector<int>(n, -1));
    dp[0][0] = grid[0][0];
    for (int i = 0; i < 2 * n - 2; i++) {
      vector<vector<int>> new_dp(n, vector<int>(n, -1));
      for (int x1 = 0; x1 <= min(i, n - 1); x1++)
        for (int x2 = 0; x2 <= min(i, n - 1); x2++) {
          if (dp[x1][x2] == -1) continue;
          for (int dx1 = 0; dx1 <= 1; dx1++)
            for (int dx2 = 0; dx2 <= 1; dx2++) {
              int nx1 = x1 + dx1, nx2 = x2 + dx2, ny1 = (i + 1) - nx1, ny2 = (i + 1) - nx2;
              if (nx1 < n && ny1 < n && nx2 < n && ny2 < n && grid[nx1][ny1] != -1 && grid[nx2][ny2] != -1) {
                if (nx1 == nx2)
                  new_dp[nx1][nx2] = max(new_dp[nx1][nx2], dp[x1][x2] + grid[nx1][ny1]);
                else
                  new_dp[nx1][nx2] = max(new_dp[nx1][nx2], dp[x1][x2] + grid[nx1][ny1] + grid[nx2][ny2]);
              }
            }
        }
      dp = new_dp;
    }
    return max(0, dp[n - 1][n - 1]);
  }
};
```