动态规划

树型dp

dp的更新以树形进行： 因为数据的依赖结构也是树形的

T285 没有上司的舞会：人之间以上司下级的关系成树的关系
每个人有高兴值，参加舞会的不能有上司关系；要高兴值和最大
上司关系 <=> 树边；所以没有上司即所选的节点之间没有树边存在

分析

- 状态表示 `dp[u][2]` ，分成两个类别
  - 【集合】 `dp[u][0]` 所有从以u为根的子树中选择，并且**不选u这个点的方案**； `dp[u][1]` 所有从以u为根的子树中选择，且**选u点的方案**
  - 【属性】max
- 状态计算：记u的子节点为s1, s2, .. sn 一般记为si；计算顺序即从叶子到根
  - 不选自己， `dp[u][0] = \sum max(dp[si][0], dp[si][1])`
  - 选自己， `dp[u][1] = \sum dp[si][0]`
  - 复杂度：dp数组有2n个状态，每次更新复杂度都与儿子数量有关，总和即树的边数，所以更新枚举的次数是$O(n-1) = O(n)$

```
const int N = 6002;
int n;
int happy[N];
int h[N], e[N], ne[N], idx;  // graph: sll storage
bool has_father[N]; // special: to tell the root

int dp[N][2];

inline void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}

void dfs(int u) {
    dp[u][1] = happy[u];
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        dfs(j);
        dp[u][0] += max(dp[j][0], dp[j][1]);
        dp[u][1] += dp[j][0];
    }
}
```

```cpp
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    /* Read in & Preprocess */
    memset(h, -1, sizeof h);
    cin >> n;
    for (int i = 1; i <= n; ++i) cin >> happy[i];
    for (int i = 0; i < n-1; ++i) {
        int a, b;
        cin >> a >> b; // a <-- b
        has_father[a] = true;
        add(b, a);
    }
    int root = 1; // Find root
    while (has_father[root]) root++;

    /* Calculate dp */
    dfs(root);

    /* Output */
    cout << max(dp[root][0], dp[root][1]) << endl;
}
```

数位dp

T338 计数问题
统计[a, b]之间的数中，按十进制看出现各个数字（0/1/2/.../9）的个数
思路：分情况讨论

- `count(n, x)`：统计1～n中x出现的次数
  - 结果为x在每一位上出现的次数之和
  - e.g 求出x=1在1～abcdefg中所有数的第4位上出现的次数，即 1 <= xxx1yyy <= abcdefg
    - (1) xxx = 000～abc-1：则yyy可以取到000～999，共abc * 1000个符合要求的数
    - (2) xxx = abc：则继续分类讨论
      - 若d < 1，则有abc1yyy 必然大于 abc0efg，yyy没得取，共0个数
      - 若d = 1，则 yyy可取000～efg，共efg+1个数
      - 若d > 1，则 yyy可取000～999，共1000个数
  - 边界情况：
    - x出现在头上，则没有情况(1)
    - x为0的时候，(1)的情况没有000的情况，因为这样指定位的0也会被算入前导零
    - x为0且在头上，此时(2)情况的第3点同样会保留前导零，这些不能算入
- 前缀和思想：题目所问即为`count(b, x) - count(a-1, x)`

复杂度：很快，单次询问中包括 10个数字 *2次count* 数范围最多8个数位 *每个数位都要计算10个数字* 分类讨论O(1) = 1600次即可，反正O(1)的级别

遇到bug：计算abc（数字序列到数值）时，搞反了，应该从高位开始加起

```
/* Calculate: num sequence -> number value */
int getNum(vector<int>& nums, int lo, int hi) {
    int res = 0;
    for (int i = hi; i >= lo; --i)
        res = res * 10 + nums[i];
    return res;
}

int qmi(int a, int k) {
    int res = 1;
    while (k) {
        if (k & 1) res *= a;
        a *= a; k >>= 1;
    }
    return res;
}

/* Count x in [1, n] */
int count(int n, int x) {
    if (n == 0) return 0;
    /* Preprocess n */
    vector<int> nums;
    for (int i = n; i; i /= 10) nums.push_back(i % 10);
    n = nums.size();

    /* Deal: 1 <= zzz'x'yyy <= abc'd'efg */
    int res = 0;
    for (int i = n-1; i >= 0; --i) {
        // case 1: zzz = 000 ~ abc-1
        if (i < n-1) {
            int zzz_cnt = getNum(nums, i+1, n-1);
            if (x == 0) zzz_cnt--;  // leading zero
            res += zzz_cnt * qmi(10, i);
        }

        // case 2: zzz = abc
        if (x == nums[i]) {         //// x == d : yyy = 000 ~ efg
            res += getNum(nums, 0, i-1) + 1;
        } else if (x < nums[i]) {   //// x <  d : yyy = 000 ~ 999
            if (i == n-1 && x == 0) continue;  // leading zero
            res += qmi(10, i);
        }
    }
    return res;
```

```
}

int main() {
    int a, b;
    while (scanf("%d%d", &a, &b), a && b) {
        if (a > b) swap(a, b);
        for (int i = 0; i <= 9; ++i) {
            printf("%d ", count(b, i) - count(a-1, i));
        }
        puts("");
    }
    return 0;
}
```

状态压缩

将某维状态看作**二进制数**，用此数表示某种方案的具体情况，数组内容则是方案数。通过将**状态压缩为整数**来达到优化转移的目的
tip: 数据范围较小时可以考虑，因为状压的状态个数不能很多，比如n=20时，$2^{20} \approx 10^6$的空间了，这个枚举量也差不多极限了

T291 蒙德里安的梦想

- 求将N*M的棋盘分割成若干1*2长方形的方案数
  思路
- 当横向方格摆完后，纵向方格就只能顺次的摆完1种方案，不会出现多余方案；所以只看横向摆即可
- `dp[i][j]`
- 状态表示：i表示列，而j则是以二进制形式存储第i列上放了横向方格(记横方格定位在**右侧方格处**)的情况
- 状态计算：从i-1转移到i的过程方案依然成立(记摆放情况为k和j)，需满足以下条件
  - 两处的摆放没有重叠，即 `(j & k) == 0`
  - i-1处留出的纵向格子数必须为偶数，否则无法填纵向格子，即 `j | k`不存在连续奇数个0
  - 满足条件后即有 `dp[i][j] = dp[i][j] + dp[i-1][k]`
- 复杂度：状态数 = 最大格子数11 * 2^{11}；转移枚举数量2^{11} ；所以总共约 4×10^7，不会超
  实现：首列为0，因为方格右侧肯定不能出现在首列

```
const int N = 12, M = 1 << N;
typedef long long ll;

int n, m;
ll dp[N][M];
bool st[N];  // preprocess for continuous zero cnt
```

```
int main() {
    int n, m;
    while (scanf("%d%d", &n, &m), n && m) {
        /* Preprocess */
        memset(dp, 0, sizeof dp);
        for (int i = 0; i < 1<<n; ++i) {
            st[i] = true; int cnt = 0;
            for (int j = 0; j < n; ++j) {
                if (i >> j & 1) {
                    if (cnt & 1) { st[i] = false; break; }
                    cnt = 0;
                } else cnt++;
            }
            if (cnt & 1) st[i] = false;
        }

        /* Calculation */
        dp[0][0] = 1;
        for (int i = 1; i <= m; ++i) {
            for (int j = 0; j < 1<<n; ++j) {
                for (int k = 0; k < 1<<n; ++k) {
                    if ((j & k) == 0 && st[j | k])
                        dp[i][j] += dp[i-1][k];
                }
            }
        }
        /* Out */
        printf("%lld\n", dp[m][0]);  // list m should all be zero
    }
}
```

T91 最短哈密顿通路：找从0走到n-1的最短哈密顿通路
思路

- 还是用整数表示状态
- `dp[i][j]`
  - 状态表示：【集合】从0走到j，**走过的所有点是**i的所有路径【属性】min
    - 状态压缩到 `i` 中：一个二进制数，每一位对应一个点，每一位表示当前这个点是否走过
  - 状态计算：对于 `dp[i][j]`，按倒数第二个点进行分类[0],[1],...[n-1]，记为k。
    - 路径为0 $\longrightarrow k \rightarrow j$，则有 `dp[i][j] = min(\forall k: dp[i-{j}, k] + a[k][j])`

```
const int N = 20, M = 1 << N;

int n;
int w[N][N];
```

```
int dp[M][N];

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            scanf("%d", &w[i][j]);

    memset(dp, 0x3f, sizeof dp);
    dp[1][0] = 0;
    for (int i = 0; i < 1<<n; ++i) {
        for (int j = 0; j < n; ++j) {
            // to be legal: bit j should be 1
            if (i >> j & 1) {
                for (int k = 0; k < n; ++k) {
                    int k_status = i - (1 << j);
                    if (k_status >> k & 1) // to be legal: after exclude bit
j, bit k should be 1
                        dp[i][j] = min(dp[i][j], dp[k_status][k] + w[k][j]);
                }
            }
        }
    }
    printf("%d\n", dp[(1<<n)-1][n-1]);
}
```

记忆化搜索

带备忘录的递归

- 有些题目难以找到dp计算的路径，这样递归反而更佳
- 空间复杂度占优

T901滑雪：在二维矩阵表示的二维平面内，找最长的滑雪轨迹。合法轨迹得是降序的数列

- 状态表示 `dp[i][j]`
  - 【集合】所有从(i, j)开始滑的路径
  - 【属性】长度max
- 状态计算：按(i, j)第一步是向何处滑的进行分类，即有4类
  - 例如：向右滑，则路径为(i, j) → (i, j+1) → end，则就有 `dp[i][j] = dp[i][j+1] + 1`；向其他方向也是一样，只不过并非都存在
  - 不可能出现换，肯定是拓扑图，因为要求是高度递减
    换一种实现方式，用递归写；话说正常动态规划的迭代，似乎计算顺序不好整

```
const int N = 310;
int n, m;
int h[N][N];
```

```
int dp[N][N];

// up right down left
const int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};
inline bool inbound(int x, int y) {
    return 1 <= x && x <= n && 1 <= y && y <= m;
}

int dpRecursive(int x, int y) {
    int& v = dp[x][y];
    if (v != 0) return v;

    int a, b;
    for (int i = 0; i < 4; ++i) {
        a = x + dx[i], b = y + dy[i];
        if (inbound(a, b) && h[a][b] < h[x][y])
            v = max(v, dpRecursive(a, b) + 1);
    }
    return v = max(v, 1);
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            scanf("%d", &h[i][j]);
    int res = 0;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            res = max(res, dpRecursive(i, j));
    printf("%d\n", res);
}
```

线性dp

LIS最长上升子序列

基本思路

- 定义 `dp[i]` 为考虑前 `i` 个元素，以第 `i` 个数字结尾的最长上升子序列的长度，注意 `nums[i]` 必须被选取
- 我们从小到大计算 dp 数组的值，在计算 dp[i] 之前，我们已经计算出 dp[0…i−1] 的值，则状态转移方程为： `dp[i]=max(dp[j])+1` ，其中 `0≤j<i` 且 `num[j]<num[i]`

```
int mx = 1;  // 找出所计算的f[i]之中的最大值，边算边找
    for (int i = 0; i < n; i++) {
        f[i] = 1;    // 设f[i]默认为1，找不到前面数字小于自己的时候就为1
        for (int j = 0; j < i; j++) {
```

```
            if (w[i] > w[j]) f[i] = max(f[i], f[j] + 1);
            // 前一个小于自己的数结尾的最大上升子序列加上自己，即+1
        }
        mx = max(mx, f[i]); // 目标
    }
```

【变种】合唱队，双向

- N位同学站成一排，音乐老师要请其中的 (N-K)位同学出列，使得剩下的 K 位同学排成合唱队形。
- 合唱对象是单峰
- 即求最长单峰序列

```
int dp_left[MAXN], dp_right[MAXN];
int height[MAXN];
int N;

int main() {
    scanf("%d", &N);
    for (int i = 0; i < N; ++i) scanf("%d", &height[i]);

    for (int i = 0; i < N; ++i) {
        dp_left[i] = 1;
        for (int j = 0; j < i; ++j) {
            if (height[i] > height[j]) {
                dp_left[i] = max(dp_left[i], dp_left[j] + 1);
            }
        }
    }
    for (int i = N-1; i >= 0; --i) {
        dp_right[i] = 1;
        for (int j = N-1; j > i; --j) {
            if (height[i] > height[j]) {
                dp_right[i] = max(dp_right[i], dp_right[j] + 1);
            }
        }
    }

    int max_student = 0;
    for (int i = 0; i < N; ++i) max_student = max(max_student, dp_left[i] +
dp_right[i] - 1);
    printf("%d\n", N - max_student);
}
```

【反链定理】
T1010 拦截导弹：问最少多少套可以拦截（每套只能降序的拦截）
第一问即LIS，第二问要求用最少的LIS覆盖全序列（贪心）

- 贪心过程：从前向后扫描每个数
    - 若现有子序列的结尾都小于当前数，则需要创建新子序列
    - 反之，则将当前数放到 **结尾大于它的子序列** 中 **结尾最小的子序列** （启发式理由：尽可能使子序列的结尾大，这样可接更多的）
- 贪心证明：当前贪心算法记为A，最优算法记为OPT
    - 首先必然有OPT $\le$ A
    - 对于每个不一样的地方，可以调整A的接法为OPT(不断交换后缀)而不使子序列数增加，所以有OPT≥A （调整法）
    - 所以得证
- 所以做法就暗合LIS的贪心二分解法： 可发现对偶问题

最少用多少个**最长非下降子序列**覆盖（宽度） $\Longleftrightarrow$ 最长**上升子序列**长度 【对偶问题】
反链定理: Dilworth定理，集合划分为数目最少的链来量化地描述任何有限偏序集的**宽度**

- 包含元素最多反链的元素数 等于 包含链数最少的链分解的链数

按行读取，非定长的数： `stringstream`，`while(cin>>xx)`
(洛谷的P1020)

```
输入: 3 4 5 4 5 2 1

直观看：对于上升子序列（每个序列单增）
3 4 5
4 5
2
1
记录上升子序列末尾的数组成数组，则为单调非增
5 5 2 1
```

```
#include <sstream>
const int MAXN = 1e5+10;
int height[MAXN];
int dp_seq[MAXN];
int n = 0;
int len = 0, cnt = 0;

void printSeq(int size) {
    printf("[dp_seq] ");
    for (int i = 0; i < size; ++i) printf("%d ", dp_seq[i]);
    printf("size = %d \n", size);
}

int main() {
    // string line;
    // getline(cin, line);
```

```
    // stringstream ssin(line);
    // while (ssin >> height[n]) n++;
    while (cin >> height[n]) n++;

    /* Question 1 */
    // Size of Longest Non-Increase Subsequence
    dp_seq[len++] = height[0];
    for (int i = 1; i < n; ++i) {
        int target = height[i];
        if (target <= dp_seq[len-1]) {
            dp_seq[len++] = target;
        } else { // target > dp_seq[len-1]
            int lo = 0, hi = len - 2, mi;
            while (lo <= hi) {
                mi = (lo + hi) >> 1;
                if (target > dp_seq[mi]) hi = mi - 1; // [tips]: has the
same direction as the current branch
                else lo = mi + 1;
            }
            dp_seq[lo] = target;
        }
    }

    /* Question 2: dual question */
    // Size of Longest Increase Subsequence <=> Minimal Nums of Longest Non-
Increase Subsequences to Cover the Whole Sequence
    dp_seq[cnt++] = height[0];
    for (int i = 1; i < n; ++i) {
        int target = height[i];
        if (target > dp_seq[cnt-1]) {
            dp_seq[cnt++] = target;
        } else { // target <= dp_seq[cnt-1]
            int lo = 0, hi = cnt - 2, mi;
            while (lo <= hi) {
                mi = (lo + hi) >> 1;
                if (target <= dp_seq[mi]) hi = mi - 1;
                else lo = mi + 1;
            }
            dp_seq[lo] = target;
        }
    }
    printf("%d\n%d\n", len, cnt);
    return 0;
}
```