动态规划

1. 背包DP

```
// 多重背包
for (int i = 1; i <= n; i++)
{
    int q = a[i].m;
    for (int j = 1; q; j *= 2)
    {
        if (j > q)
        {
            j = q;
        }
        q -= j;
        for (int k = w; k >= j * a[i].w; k--)
        {
            f[k] = max(f[k], f[k - j * a[i].w] + j * a[i].v);
        }
    }
}
```

二进制分组优化

```
index = 0;
for (int i = 1; i <= m; i++)
{
    int c = 1, p, h, k;
    cin >> p >> h >> k;
    while (k > c)
    {
        k -= c;
        list[++index].w = c * p;
        list[index].v = c * h;
        c *= 2;
    }
    list[++index].w = p * k;
    list[index].v = h * k;
}
```

2. 状态压缩DP

```
int cnt[1024];
int dp[40][1024][90];
int can[2000], num = 0;
int S = 1 << n;
for (int s = 0; s < S; s++)
{
```

```
    if ((s << 1) & s)
    {
        continue;
    }
    can[++num] = s;
    for (int j = 0; j < n; j++)
    {
        if ((s >> j) & 1)
        {
            cnt[num]++;
        }
    }
    dp[1][num][cnt[num]] = 1;
}
for (int i = 2; i <= n; i++)
{
    for (int j = 1; j <= num; j++)
    {
        int x = can[j];
        for (int p = 1; p <= num; p++)
        {
            int y = can[p];
            if ((y & x) || ((y << 1) & x) || ((y >> 1) & x))
                continue;
            for (int l = 0; l <= k; l++)
            {
                dp[i][j][cnt[j] + l] += dp[i - 1][p][l];
            }
        }
    }
}
```

3. 数位DP

现在问有多少的数比12345小

1.关于前导0：00999→999，09999→9999

2.关于limit前面的数是否紧贴上限

如果前面的数是紧贴上限的，当前这位枚举的上限便是当前数的上限

如果前面的数不是紧贴上限的，当前这位枚举的上限便是 9

3.关于DP维度

一般来说，DFS有几个状态，DP就几个维度

比如现在DP就是DP [pos] [limt] [zero]

4.关于记忆化DP

现在枚举到了 10××× 和 11×××

显然 这两种状态后面的×××状态数是一样的

重点：dp[pos][limit][zero]表示前面的数枚举状态确定，后面的数有多少种可能

5.关于DP细节

一般来说我们一开始都memset(dp,-1,sizeof(dp))

如果dp[pos][limt][zero]!=-1 return dp[pos][limit][zero];

6.关于初始化：

一开始 limit 是1，表示一开始的数只能选 1~a[1]

一开始zero 是1，假定表示前面的数全为0

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define mp make_pair
#define pb push_back  // vector函数
#define popb pop_back // vector函数
#define fi first
#define se second
const int N = 20;
// const int M=;
// const int inf=0x3f3f3f3f;     //一般为int赋最大值,不用于memset中
// const ll INF=0x3fffffffffff; //一般为ll赋最大值,不用于memset中
int T, n, len, a[N], dp[N][2][2];
inline int read()
{
    int x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9')
    {
        if (ch == '-')
            f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9')
    {
        x = (x << 1) + (x << 3) + (ch ^ 48);
        ch = getchar();
    }
    return x * f;
}
int dfs(int pos, bool lim, bool zero)
{
    if (pos > len)
        return 1;
    if (dp[pos][lim][zero] != -1)
        return dp[pos][lim][zero];
    int res = 0, num = lim ? a[pos] : 9;
    for (int i = 0; i <= num; i++)
        res += dfs(pos + 1, lim && i == num, zero && i == 0);
    return dp[pos][lim][zero] = res;
```

```
}
int solve(int x)
{
    len = 0;
    memset(dp, -1, sizeof(dp));
    for (; x; x /= 10)
        a[++len] = x % 10;
    reverse(a + 1, a + len + 1);
    return dfs(1, 1, 1);
}
int main()
{
    int l = read(), r = read();
    printf("%d\n", solve(r) - solve(l - 1));
    return 0;
}
```