9.dp入门

1、实现斐波那契数列

```
#include <iostream>
using namespace std;
const int N = 1e3;
typedef long long ll;
ll f[N];

int main()
{
  int n;
  cin >> n;
  f[0] = f[1] = 1;
  for (int i = 2; i <= n; i++)
  {
    f[i] = f[i-1] + f[i-2];
  }
  cout << f[n];
  return 0;
}
```

2、二维递推

```
#include <iostream>
using namespace std;
const int N = 55;
typedef long long ll;
ll f[N][N];

void init()
{
    for (int i = 1; i < N;i++)
    {
        for (int j = 1; j <= i;j++)
        {
            if(j==1 || j==i)
                f[i][j] = 1;
            else
                f[i][j] = f[i-1][j-1] + f[i-1][j];
        }
    }
}
```

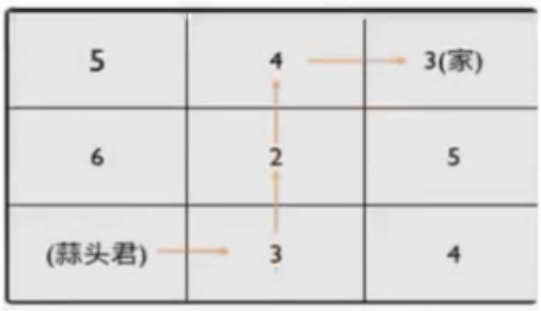
```

int main()
{
  init();
  int n,m;
  cin >> n >> m;
  cout << f[n][m] << endl;
  return 0;
}
```

3、实现马踏过河卒

```
#include <iostream>
using namespace std;
int dx[8] = {1,1,2,2,-1,-1,-2,-2};
int dy[8] = {2,-2,1,-1,2,-2,1,-1};
bool d[30][30];
long long dp[30][30];
int main()
{
  int n,m,cx,cy;
  cin >> n >> m >> cx >> cy;
  d[cx][cy] = true;
  for (int i = 1; i < 8 ;i++ )
  {
    int nx = cx + dx[i];
    int ny = cy + dy[i];
    if (nx >= 0 && nx <= n && ny >=0 && ny <=m)
    {
        d[nx][ny] = true;
    }
  }
  dp[0][0] = 1;
  for (int i = 0; i <= n; i++)
  {
    for (int j = 0; j <= m; j++)
    {
        if (d[i][j] == false)
        {
            if(i)
                dp[i][j] += dp[i-1][j];
            if (j)
                dp[i][j] += dp[i][j-1];
        }
    }
  }
  cout << dp[n][m] <<endl;
  return 0;
}
```

状态转移方程：动态规划中本阶段的状态往往是上一阶段的状态和上一阶段的决策的结果，由第i段的状态f(i) ,和决策u(i)来确定第i＋1段的状态。状态转移表示为F(i＋1)=T(f(i), u(i))，称为状态转移方程。
同时，需要注意便捷点的位置

4、蒜头君回家




```
for (int i = 1; i <= n; i++)
{
  for (int j = 1; j <= n; j++)
  {
    if (i==1&&j==1)
      continue;
    else if ( i == 1)
      dp[i][j] = dp[i][j-1] + a[i][j];
    else if ( j == 1)
      dp[i][j] = dp[i-1][j] + a[i][j];
    else
      dp[i][j] = min(dp[i-1][j],dp[i][j-1]) + a[i][j];
  }
}
```

5、实现捡水果

```
#include <iostream>
using namespace std;
const int N = 1e3 + 9;
const int inf = 10000000000;
int f[N][N];
int main()
{
  int n,ans=-inf;
  cin >> n;
  for (int i = 1; i <= n; i++)
  {
    for (int j = 1; j <= i; j++)
    {
        cin >> f[i][j];
    }
  }
  for (int i = 1; i <= n; i++)
  {
```

```
    for (int j = 1; j <= i; j++)
    {
        f[i][j] += max(f[i-1][j],f[i-1][j-1]);
        if ( i== n)
            ans = max(f[i][j],ans);
    }
  }
  if(ans == -inf)
    ans = 0;
  cout << ans <<endl;
  return 0;
}
```

6、多维状态转移方程

```
#include <iostream>
using namespace std;
const int N = 1e2 + 9;
const int inf = 100000000;

int f[N][N][N];

int main()
{
    int x,y,z;
    cin >> x >>y >> z;
    for (int i = 0; i <= x; i++)
    {
        for (int j = 0; j <= y;j++)
        {
            for (int k = 0; k <= z;k++)
            {
                cin >> f[i][j][k];
            }
        }
    }
    for (int i = 0; i <= x; i++)
    {
        for (int j = 0; j <= y;j++)
        {
            for (int k = 0; k <= z;k++)
            {
                int mi = inf;
                if (i!= 0)
                {
                    mi = min(mi,f[i-1][j][k]);
                }
                if (j!= 0)
                {
                    mi = min(mi,f[i][j-1][k]);
                }
```

```
                if (k!= 0)
                {
                    mi = min(mi,f[i][j][k-1]);
                }
                if (mi != inf)
                {
                    f[i][j][k] += mi;
                }

            }
        }
    }
    cout << f[x][y][z];
    return 0;
}
```

7、弹簧板

有一个小球掉落在一串连续的弹簧板上，小球落到某一个弹簧板后，会被弹到某一个地点，直到小球被弹到弹簧板以外的地方。

假设有 n 个连续的弹簧板，每个弹簧板占一个单位距离，$a[i]$ 代表代表第 i 个弹簧板会把小球向前弹 $a[i]$ 个距离。比如位置 1 的弹簧能让小球前进 2 个距离到达位置 3。如果小球落到某个弹簧板后，经过一系列弹跳会被弹出弹簧板，那么小球就能从这个弹簧板弹出来。现在希望你计算出小球从任意一个弹簧板落下，最多会被弹多少次后，才会弹出弹簧板。

输入格式

第一个行输入一个 n 代表一共有 $n(1 \leq n \leq 100000)$ 个弹簧板。第二行输入 n 个数字，中间用空格分开。第 i 个数字 $a_i(1 \leq a_i \leq 30)$ 代表第 i 个弹簧板可以让小球移动的距离。

输出格式

输出一个整数，代表小球最多经过多少次才能弹出弹簧板。

```
for (int i = 1; i <= n ; i++)
{
  cin >> a[i];
}
int ans = 0;

for (int i = n; i >= 1; i--)
{
  dp[i] = dp[i+a[i]] + 1;
  ans = max(dp[i],ans);
```

```
}
```

8、蒜头君的新游戏

工作空闲之余，蒜头君经常带着同事们做游戏，最近蒜头君发明了一个好玩的新游戏：n 位同事围成一个圈，同事 A 手里拿着一个兔妮妮的娃娃。蒜头君喊游戏开始，每位手里拿着娃娃的同事可以选择将娃娃传给左边或者右边的同学，当蒜头君喊游戏结束时，停止传娃娃。此时手里拿着娃娃的同事即是败者。

玩了几轮之后，蒜头君想到一个问题：有多少种不同的方法，使得从同事 A 开始传娃娃，传了 m 次之后又回到了同事 A 手里。两种方法，如果接娃娃的同事不同，或者接娃娃的顺序不同均视为不同的方法。例如 $1->2->3->1$ 和 $1->3->2->1$ 是两种不同的方法。

输入格式

输入一行，输入两个整数 $n, m(3 \leq n \leq 30, 1 \leq m \leq 30)$，表示一共有 n 位同事一起游戏，一共传 m 次娃娃。

输出格式

输出一行，输出一个整数，表示一共有多少种不同的传娃娃方法。

```
int f[35][35]; //传了多少次，现在在谁的手上

f[0][1] = 1;
for (int i = 1; i <= m; i ++)
{
  for (int j = 1; j <= n; j++)
  {
    if (j == 1)  //第1个人
    {
      f[i][j] = f[i-1][2] + f[i-1][n];
    }
    else if ( j == n)
    {
      f[i][j] = f[i-1][1] + f[i-1][n-1];
    }
    else
    {
      f[i][j] = f[i-1][j-1] + f[i-1][j+1];
    }
  }
}
cout << f[m][1] <<endl;
```