##### 方法二

我们以层来划分，显然，第i层是由第i－1层而来。设f[i]表示从起点出发，要走i步的方案数。则有：

f[i]＝f[i－1]（从起点向上走一步后，再走i－1步）

f[i]＝f[i－2]＋f[i－3]＋…＋f[0]＋f[-1] （从起点向左走1，2，…，i－1步后再向上。其中f[-1]表示停在最左边，不再往上走，其实此处省略不影响结果。）

f[i]＝f[i－2]＋f[i－3]＋…＋f[0]＋f[-1] （向右走1，2，3，…，i－1步后向上。其中f[-1]表示停在最右边，不再往上走，其实此处省略不影响结果。）

三数相加即为结果。

参考程序如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | //布阵 — 方法2  #include <bits/stdc++.h>  using namespace std;  unsigned long long f[35]= {1,1};  int main()  {  int n;  scanf("%d",&n);  for(int i=2; i<=n+1; i++)  {  f[i]=f[i-1];  for(int j=1; j<=i; j++)  f[i]+=2\*f[i-j-1];  }  printf("%llu\n",f[n+1]);  return 0;  } |

##### 方法三

设从i点可以向上、左、右走的步数记作f1[i]，只可以向上、左走的步数记为f2[i]，只可以向上、右走的步数记作f3[i]，则按递归的思路，显然有：

f1[i]＝f1[i－1]＋f2[i－1]＋f3[i－1]

f2[i]＝f1[i－1]＋f2[i－1]

f3[i]＝f1[i－1]＋f3[i－1]

由于f2和f3显然相同，故可简化为只要f2。

参考程序如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | //布阵 — 方法3  #include <bits/stdc++.h>  using namespace std;  long long f1[35]= {1},f2[35]= {1};  int main()  {  int n;  scanf("%d",&n);  for(int i=1; i<=n+1; i++)  {  f1[i]=f1[i-1]+2\*f2[i-1];  f2[i]=f1[i-1]+f2[i-1];  }  printf("%lld\n",f1[n]);  return 0;  } |

|  |  |
| --- | --- |
|  | 该方法用的递推式其实还可以继续化简。 |

##### 方法四

令f[i]为共走i步的走法数量，令f↑[i]为共走i步但最后一步是向上的走法数量，同理定义f←[i]和f→[i]。那么有：

f[i]＝f↑[i]＋f←[i]＋f→[i] 　　　　　　　　　　　（1）

若某一步向上走，那么下一步往任意一个方向都可以，共3种方法。

若某一步向左走，那么下一步只能向上或者向左，共2种方法。

若某一步向右走，那么下一步只能向上或者向右，共2种方法。

也就是说下一步的走法数量f[i＋1]＝3×f↑[i]＋2×f←[i]＋2×f→[i]。

将（1）代入上式可得：

f[i＋1]＝f↑[i]＋2×f[i]。 　　　　　　　　　　　　（2）

另外，若某一步向上走，那么上一步往任意一个方向走都是有可能的，即：

f↑[i]＝f↑[i－1]＋f←[i－1]＋f→[i－1] 　　　　　 （3）

将（1），（3）代入（2）后就得到了f[i＋1]＝f[i－1]＋2×f[i]。

转化状态转移方程f[i]＝2×f[i－1]＋f[i－2]。

边界为f[0]＝1，f[1]＝3。

参考程序如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | //布阵 — 方法4  #include <bits/stdc++.h>  using namespace std;  int main()  {  long long n,f[2]= {1,3}; //采用滚动数组  scanf("%lld",&n);  for(int i=2; i<=n; i++)  f[i%2]=2\*f[(i-1)%2]+f[(i-2)%2];  printf("%lld\n",f[n%2]);  return 0;  } |