**Unity3D教程：游戏开发算法（一）**

Posted on 2013年06月25日 by U3d / [Unity3D 基础教程](http://www.unitymanual.com/category/manual/unity3d-%e5%9f%ba%e7%a1%80%e6%95%99%e7%a8%8b)/被围观 38 次

要使计算机能完成人们预定的工作，首先必须为如何完成预定的工作设计一个算法，然后再根据算法编写程序。计算机程序要对问题的每个对象和处理规则给出正确详尽的描述，其中程序的数据结构和变量用来描述问题的对象，程序结构、函数和语句用来描述问题的算法。算法数据结构是程序的两个重要方面。

算法是问题求解过程的精确描述，一个算法由有限条可完全机械地执行的、有确定结果的指令组成。指令正确地描述了要完成的任务和它们被执行的顺序。计算机按算法指令所描述的顺序执行算法的指令能在有限的步骤内终止，或终止于给出问题的解，或终止于指出问题对此输入数据无解。

通常求解一个问题可能会有多种算法可供选择，选择的主要标准是算法的正确性和可靠性，简单性和易理解性。其次是算法所需要的存储空间少和执行更快等。

算法设计是一件非常困难的工作，经常采用的算法设计技术主要有迭代法、穷举搜索法、递推法、贪婪法、回溯法、分治法、动态规划法等等。另外，为了更简洁的形式设计和藐视算法，在算法设计时又常常采用递归技术，用递归描述算法。

一、迭代法

迭代法是用于求方程或方程组近似根的一种常用的算法设计方法。设方程为f(x)=0，用某种数学方法导出等价的形式x=g(x)，然后按以下步骤执行：

（1） 选一个方程的近似根，赋给变量x0；

（2） 将x0的值保存于变量x1，然后计算g(x1)，并将结果存于变量x0；

（3） 当x0与x1的差的绝对值还小于指定的精度要求时，重复步骤（2）的计算。

若方程有根，并且用上述方法计算出来的近似根序列收敛，则按上述方法求得的x0就认为是方程的根。上述算法用C程序的形式表示为：

【算法】迭代法求方程的根

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | { x0=初始近似根； |
| 02 |  |
| 03 | **do** { |
| 04 |  |
| 05 | x1=x0； |
| 06 |  |
| 07 | x0=g(x1)； */\*按特定的方程计算新的近似根\*/* |
| 08 |  |
| 09 | } **while** ( fabs(x0-x1)>Epsilon)； |
| 10 |  |
| 11 | printf(“方程的近似根是%f\n”，x0)； |
| 12 |  |
| 13 | } |

迭代算法也常用于求方程组的根，令X=（x0，x1，…，xn-1），设方程组为：xi=gi(X) (I=0，1，…，n-1)，则求方程组根的迭代算法可描述如下：

【算法】迭代法求方程组的根

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | { **for** (i=0;i x=初始近似根; |
| 02 |  |
| 03 | **do** { |
| 04 |  |
| 05 | **for** (i=0;i y=x; |
| 06 |  |
| 07 | **for** (i=0;i x=gi(X); |
| 08 |  |
| 09 | **for** (delta=0.0,i=0;i **if** (fabs(y-x)>delta) delta=fabs(y-x)； |
| 10 |  |
| 11 | } **while** (delta>Epsilon)； |
| 12 |  |
| 13 | **for** (i=0;i printf(“变量x[%d]的近似根是 %f”，I，x)； |
| 14 |  |
| 15 | printf(“\n”)； |
| 16 |  |
| 17 | } |
| 18 |  |

具体使用迭代法求根时应注意以下两种可能发生的情况：

（1） 如果方程无解，算法求出的近似根序列就不会收敛，迭代过程会变成死循环，因此在使用迭代算法前应先考察方程是否有解，并在程序中对迭代的次数给予限制；

（2） 方程虽然有解，但迭代公式选择不当，或迭代的初始近似根选择不合理，也会导致迭代失败。

二、穷举搜索法

穷举搜索法是对可能是解的众多候选解按某种顺序进行逐一枚举和检验，并从众找出那些符合要求的候选解作为问题的解。

【问题】 将A、B、C、D、E、F这六个变量排成如图所示的三角形，这六个变量分别取[1，6]上的整数，且均不相同。求使三角形三条边上的变量之和相等的全部解。如图就是一个解。

程序引入变量a、b、c、d、e、f，并让它们分别顺序取1至6的证书，在它们互不相同的条件下，测试由它们排成的如图所示的三角形三条边上的变量之和是否相等，如相等即为一种满足要求的排列，把它们输出。当这些变量取尽所有的组合后，程序就可得到全部可能的解。细节见下面的程序。

【程序1】

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | # include |
| 02 |  |
| 03 | **void** main() |
| 04 |  |
| 05 | { **int** a,b,c,d,e,f; |
| 06 |  |
| 07 | **for** (a=1;a<=6;a++) |
| 08 |  |
| 09 | **for** (b=1;b<=6;b++) { |
| 10 |  |
| 11 | **if** (b==a) **continue**; |
| 12 |  |
| 13 | **for** (c=1;c<=6;c++) { |
| 14 |  |
| 15 | **if** (c==a)||(c==b) **continue**; |
| 16 |  |
| 17 | **for** (d=1;d<=6;d++) { |
| 18 |  |
| 19 | **if** (d==a)||(d==b)||(d==c) **continue**; |
| 20 |  |
| 21 | **for** (e=1;e<=6;e++) { |
| 22 |  |
| 23 | **if** (e==a)||(e==b)||(e==c)||(e==d) **continue**; |
| 24 |  |
| 25 | f=21-(a+b+c+d+e); |
| 26 |  |
| 27 | **if** ((a+b+c==c+d+e))&&(a+b+c==e+f+a)) { |
| 28 |  |
| 29 | printf(“%6d,a); |
| 30 |  |
| 31 | printf(”%4d%4d“,b,f); |
| 32 |  |
| 33 | printf(”%2d%4d%4d“,c,d,e); |
| 34 |  |
| 35 | scanf(”%\*c“); |
| 36 |  |
| 37 | } |
| 38 |  |
| 39 | } |
| 40 |  |
| 41 | } |
| 42 |  |
| 43 | } |
| 44 |  |
| 45 | } |
| 46 |  |
| 47 | } |

按穷举法编写的程序通常不能适应变化的情况。如问题改成有9个变量排成三角形，每条边有4个变量的情况，程序的循环重数就要相应改变。

对一组数穷尽所有排列，还有更直接的方法。将一个排列看作一个长整数，则所有排列对应着一组整数。将这组整数按从小到大的顺序排列排成一个整数，从对应最小的整数开始。按数列的递增顺序逐一列举每个排列对应的每个整数，这能更有效地完成排列的穷举。从一个排列找出对应数列的下一个排列可在当前排列的基础上作部分调整来实现。倘若当前排列为1，2，4，6，5，3，并令其对应的长整数为124653。要寻找比长整数124653更大的排列，可从该排列的最后一个数字顺序向前逐位考察，当发现排列中的某个数字比它前一个数字大时，如本例中的6比它的前一位数字4大，这说明还有对应更大整数的排列。但为了顺序从小到大列举出所有的排列，不能立即调整得太大，如本例中将数字6与数字4交换得到的排列126453就不是排列124653的下一个排列。为了得到排列124653的下一个排列，应从已经考察过的那部分数字中选出比数字大，但又是它们中最小的那一个数字，比如数字5，与数字4交换。该数字也是从后向前考察过程中第一个比4大的数字。5与4交换后，得到排列125643。在前面数字1，2，5固定的情况下，还应选择对应最小整数的那个排列，为此还需将后面那部分数字的排列顺序颠倒，如将数字6，4，3的排列顺序颠倒，得到排列1，2，5，3，4，6，这才是排列1，2，4，6，5，3的下一个排列。按以上想法编写的程序如下。

【程序2】

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | # include |
| 02 |  |
| 03 | # define SIDE\_N 3 |
| 04 |  |
| 05 | # define LENGTH 3 |
| 06 |  |
| 07 | # define VARIABLES 6 |
| 08 |  |
| 09 | **int** A,B,C,D,E,F; |
| 10 |  |
| 11 | **int** \*pt[]={&A,&B,&C,&D,&E,&F}; |
| 12 |  |
| 13 | **int** \*side[SIDE\_N][LENGTH]={&A,&B,&C,&C,&D,&E,&E,&F,&A}; |
| 14 |  |
| 15 | **int** side\_total[SIDE\_N]; |
| 16 |  |
| 17 | main{} |
| 18 |  |
| 19 | { **int** i,j,t,equal; |
| 20 |  |
| 21 | **for** (j=0;j　　 \*pt[j]=j+1; |
| 22 |  |
| 23 | **while**(1) |
| 24 |  |
| 25 | { **for** (i=0;i　　 { **for** (t=j=0;j　　 t+=\*side[i][j]; |
| 26 |  |
| 27 | side\_total[i]=t; |
| 28 |  |
| 29 | } |
| 30 |  |
| 31 | **for** (equal=1,i=0;equal&&i　　 **if** (side\_total[i]!=side\_total[i+1] equal=0; |
| 32 |  |
| 33 | **if** (equal) |
| 34 |  |
| 35 | { **for** (i=1;i　　 printf(”%4d“,\*pt[i]); |
| 36 |  |
| 37 | printf(”\n“); |
| 38 |  |
| 39 | scanf(”%\*c“); |
| 40 |  |
| 41 | } |
| 42 |  |
| 43 | **for** (j=VARIABLES-1;j>0;j--) |
| 44 |  |
| 45 | **if** (\*pt[j]>\*pt[j-1]) **break**; |
| 46 |  |
| 47 | **if** (j==0) **break**; |
| 48 |  |
| 49 | **for** (i=VARIABLES-1;i>=j;i--) |
| 50 |  |
| 51 | **if** (\*pt[i]>\*pt[i-1]) **break**; |
| 52 |  |
| 53 | t=\*pt[j-1];\* pt[j-1] =\* pt[i]; \*pt[i]=t; |
| 54 |  |
| 55 | **for** (i=VARIABLES-1;i>j;i--,j++) |
| 56 |  |
| 57 | { t=\*pt[j]; \*pt[j] =\* pt[i]; \*pt[i]=t; } |
| 58 |  |
| 59 | } |
| 60 |  |
| 61 | } |

从上述问题解决的方法中，最重要的因素就是确定某种方法来确定所有的候选解。下面再用一个示例来加以说明。

【问题】 背包问题

问题描述：有不同价值、不同重量的物品n件，求从这n件物品中选取一部分物品的选择方案，使选中物品的总重量不超过指定的限制重量，但选中物品的价值之和最大。//Unity3D教程手册：www.unitymanual.com

设n个物品的重量和价值分别存储于数组w[ ]和v[ ]中，限制重量为tw。考虑一个n元组（x0，x1，…，xn-1），其中xi=0 表示第i个物品没有选取，而xi=1则表示第i个物品被选取。显然这个n元组等价于一个选择方案。用枚举法解决背包问题，需要枚举所有的选取方案，而根据上述方法，我们只要枚举所有的n元组，就可以得到问题的解。

显然，每个分量取值为0或1的n元组的个数共为2n个。而每个n元组其实对应了一个长度为n的二进制数，且这些二进制数的取值范围为0～2n-1。因此，如果把0～2n-1分别转化为相应的二进制数，则可以得到我们所需要的2n个n元组。

【算法】

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | maxv=0; |
| 02 |  |
| 03 | **for** (i=0;i<2n;i++) |
| 04 |  |
| 05 | { B[0..n-1]=0; |
| 06 |  |
| 07 | 把i转化为二进制数，存储于数组B中; |
| 08 |  |
| 09 | temp\_w=0; |
| 10 |  |
| 11 | temp\_v=0; |
| 12 |  |
| 13 | **for** (j=0;j　　 { **if** (B[j]==1) |
| 14 |  |
| 15 | { temp\_w=temp\_w+w[j]; |
| 16 |  |
| 17 | temp\_v=temp\_v+v[j]; |
| 18 |  |
| 19 | } |
| 20 |  |
| 21 | **if** ((temp\_w<=tw)&&(temp\_v>maxv)) |
| 22 |  |
| 23 | { maxv=temp\_v; |
| 24 |  |
| 25 | 保存该B数组； |
| 26 |  |
| 27 | } |
| 28 |  |
| 29 | } |
| 30 |  |
| 31 | } |
| 32 |  |

三、递推法

递推法是利用问题本身所具有的一种递推关系求问题解的一种方法。设要求问题规模为N的解，当N=1时，解或为已知，或能非常方便地得到解。能采用递推法构造算法的问题有重要的递推性质，即当得到问题规模为i-1的解后，由问题的递推性质，能从已求得的规模为1，2，…，i-1的一系列解，构造出问题规模为I的解。这样，程序可从i=0或i=1出发，重复地，由已知至i-1规模的解，通过递推，获得规模为i的解，直至得到规模为N的解。//Unity3D教程手册：www.unitymanual.com

【问题】 阶乘计算

问题描述：编写程序，对给定的n（n≦100），计算并输出k的阶乘k！（k=1，2，…，n）的全部有效数字。

由于要求的整数可能大大超出一般整数的位数，程序用一维数组存储长整数，存储长整数数组的每个元素只存储长整数的一位数字。如有m位成整数N用数组a[ ]存储：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | N=a[m]×10m-1+a[m-1]×10m-2+ … +a[2]×101+a[1]×100 |

并用a[0]存储长整数N的位数m，即a[0]=m。按上述约定，数组的每个元素存储k的阶乘k！的一位数字，并从低位到高位依次存于数组的第二个元素、第三个元素……。例如，5！=120，在数组中的存储形式为：

3 0 2 1 ……

首元素3表示长整数是一个3位数，接着是低位到高位依次是0、2、1，表示成整数120。

计算阶乘k！可采用对已求得的阶乘(k-1)！连续累加k-1次后求得。例如，已知4！=24，计算5！，可对原来的24累加4次24后得到120。细节见以下程序。

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | # include |
| 02 |  |
| 03 | # include |
| 04 |  |
| 05 | # define MAXN 1000 |
| 06 |  |
| 07 | **void** pnext(**int** a[ ],**int** k) |
| 08 |  |
| 09 | { **int** \*b,m=a[0],i,j,r,carry; |
| 10 |  |
| 11 | b=(**int** \* ) malloc(sizeof(**int**)\* (m+1)); |
| 12 |  |
| 13 | **for** ( i=1;i<=m;i++) b[i]=a[i]; |
| 14 |  |
| 15 | **for** ( j=1;j<=k;j++) |
| 16 |  |
| 17 | { **for** ( carry=0,i=1;i<=m;i++) |
| 18 |  |
| 19 | { r=(i　　 a[i]=r%10; |
| 20 |  |
| 21 | carry=r/10; |
| 22 |  |
| 23 | } |
| 24 |  |
| 25 | **if** (carry) a[++m]=carry; |
| 26 |  |
| 27 | } |
| 28 |  |
| 29 | free(b); |
| 30 |  |
| 31 | a[0]=m; |
| 32 |  |
| 33 | } |
| 34 |  |
| 35 | **void** write(**int** \*a,**int** k) |
| 36 |  |
| 37 | { **int** i; |
| 38 |  |
| 39 | printf(”%4d！=“,k); |
| 40 |  |
| 41 | **for** (i=a[0];i>0;i--) |
| 42 |  |
| 43 | printf(”%d“,a[i]); |
| 44 |  |
| 45 | printf(”\n\n“); |
| 46 |  |
| 47 | } |
| 48 |  |
| 49 | **void** main() |
| 50 |  |
| 51 | { **int** a[MAXN],n,k; |
| 52 |  |
| 53 | printf(”Enter the number n: “); |
| 54 |  |
| 55 | scanf(”%d“,&n); |
| 56 |  |
| 57 | a[0]=1; |
| 58 |  |
| 59 | a[1]=1; |
| 60 |  |
| 61 | write(a,1); |
| 62 |  |
| 63 | **for** (k=2;k<=n;k++) |
| 64 |  |
| 65 | { pnext(a,k); |
| 66 |  |
| 67 | write(a,k); |
| 68 |  |
| 69 | getchar(); |
| 70 |  |
| 71 | } |
| 72 |  |
| 73 | } |