1. 时针分针重合几次

表面上有60个小格，每小格代表一分钟，

时针每分钟走1/12小格，分针每分钟走1小格，从第一次重合到第二次重合分针比时针多走一圈即60小格，所以

60/（1-1/12）=720/11

每隔720/11分才重合一次（而并不是每小时重合一次）

1440里有22个720/11，如果说算上0点和24点，那也是重合23次而已，但我觉得0点应该算到前一天的24点头上，所以每一天循环下来重合22次啊

2. 找出字符串的最长不重复子串，输出长度

建一个256个单元的数组，每一个单元代表一个字符，数组中保存上次该字符上次出现的位置；

依次读入字符串，同时维护数组的值；

如果遇到冲突了，就返回冲突字符中保存的位置，继续第二步。也可以用hashmap保存已经出现的字符和字符的位置

3. 说是有一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出

现的前十个词。

先用哈希，统计每个词出现的次数，然后在用在N个数中找出前K大个数的方法找出出现

次数最多的前10个词。

4. 如题3，但是车次文件特别大，没有办法一次读入内存。

1) 直接排序，写文件时，同时写入字符串及其出现

次数。

2) 可以用哈希，比如先根据字符串的第一个字符将字符串换分为多个区域，每个区域的字符串写到一个文件内，然后再用哈希+堆统计每个区域内前10个频率最高的字符串，最后求出所有字符串中前10个频率最高的字符串。

5. 有一个整数n，将n分解成若干个整数之和，问如何分解能使这些数的乘积最大，输出这个乘积m。例如：n=12

（1）分解为1+1+1+…+1，12个1, m=1\*1\*1……\*1=1

（2）分解为2+2+…+2，6个2， m=64

（3）分解为3+3+3+3，4个3， m=81

（4）大于等于4时分解时只能分解为2和3，且2最多两个

f(n) = 3\*f(n-3) n>4

f(4) = 2\*2

f(3) = 3

f(2) = 2分解为4+4+4，3个4， m=64

6. 求数组n中出现次数超过一半的数

把数组分成[n/2]组，则至少有一组包含重复的数，因为如果无重复数，则最多只有出现次数等于一半的数。算法如下：

k<-n;

while k>3 do

把数组分成[k/2]组;

for i=1 to [k/2] do

    if 组内2个数相同，则任取一个数留下;

    else 2个数同时扔掉;

k<-剩下的数

if k=3

    then 任取2个数进行比较;

      if 两个数不同，则2个数都扔掉

       else 任取一个数

    if k=2 or 1 then 任取一数

7. A文件中最多有n个正整数，而且每个数均小于n，n <=10的七次方。不会出现重复的数。

要求对A文件中的数进行排序，可用内存为1M，磁盘可用空间足够。

不要把任何问题都往很复杂的算法上靠，最直接最简单的解决问题才是工程师应有的素质，

题目给的很有分寸：n个数，都小于n，两两不同，1M=10^6byte=10^7bit的内存，n <10^7

思路：

把1M内存看作是一个长度为10^7的位数组，每一位都初始化为0

从头扫描n个数，如果碰到i，就把位数组的第i个位置置为1，

1M内存有点少， （1M = 8M bits), 可以代表8M整数，现在n <=10的七次方，你可以读2遍文件，就可以完成排序了。第一次排n <8M得数， 第2遍排 8M<="" div="" style="word-wrap: break-word;">

8. 有10亿个杂乱无章的数，怎样最快地求出其中前1000大的数。

1) 建一个1000个数的堆，复杂度为N\*(log1000)=10N

2) 1.用每一个BIT标识一个整数的存在与否，这样一个字节可以标识8个整数的存在与否，对于所有32位的整数，需要512Mb，所以开辟一个512Mb的字符数组A，初始全0

   2.依次读取每个数n，将A[n>>3]设置为A[n>>3]|(1<<="" div="" style="word-wrap: break-word;">

   3.在A中，从大到小读取1000个值为1的数，就是最大的1000个数了。

这样读文件就只需要1遍，在不考虑内存开销的情况下，应该是速度最快的方法了。

9. 一棵树节点1, 2, 3, ... , n. 怎样实现：

先进行O(n)预处理，然后任给两个节点，用O(1)判断它们的父子关系

dfs一遍，记录每个结点的开始访问时间Si和结束访问时间Ei

对于两个节点i,j，若区间[Si,Ei]包含[Sj,Ej]，则i是j的祖先。给每个节点哈夫曼编码也行，但只适合一般的二叉树，而实际问题未必是Binary的，所以编码有局限性

10. 给定一个二叉树，求其中N（N>=2）个节点的最近公共祖先节点。每个节点只有左右孩

子指

针，没有父指针。

后序递归给每个节点打分，每个节点的分数=左分数+右分数+k，如果某孩子是给定节点则+1

最深的得分为N的节点就是所求吧，细节上应该不用递归结束就可以得到这个节点

11. 如何打印如下的螺旋队列：

21 22 。。。。

20 7 8 9 10

19 6 1 2 11

18 5 4 3 12

17 16 15 14 13

#include

#define max(a,b) ((a)<(b)?(b):(a))

#define abs(a) ((a)>0?(a):-(a))

int foo(int x, int y)

{

int t = max(abs(x), abs(y));

int u = t + t;

int v = u - 1;

v = v \* v + u;

if (x == -t)

    v += u + t - y;

else if (y == -t)

    v += 3 \* u + x - t;

else if (y == t )

    v += t - x;

else

          v += y - t;

return v;

}

int main()

{

int x, y;

for (y=-2;y<=2;y++)

{

    for (x=-2;x<=2;x++)

      printf("%5d", foo(x, y));

    printf("\n");

}

return 0;

}

第 0 层规定为中间的那个 1，第 1 层为 2 到 9，第 2 层为 10 到 25，……好像看出一点名堂来了？注意到 1、9、25、……不就是平方数吗？而且是连续奇数（1、3、5、……）的平方数。这些数还跟层数相关，推算一下就可以知道第 t 层之内一共有 (2t-1)^2 个数，因而第 t 层会从 [(2t-1)^2] + 1 开始继续往外螺旋。给定坐标 (x,y)，如何知道该点处于第几层？so easy，层数 t = max(|x|,|y|)。

知道了层数，接下来就好办多了，这时我们就知道所求的那点一定在第 t 层这个圈上，顺着往下数就是了。要注意的就是螺旋队列数值增长方向和坐标轴正方向并不一定相同。我们可以分成四种情况——上、下、左、右——或者——东、南、西、北，分别处于四条边上来分析。

东|右：x == t，队列增长方向和 y 轴一致，正东方向（y = 0）数值为 (2t-1)^2 + t，所以 v = (2t-1)^2 + t + y

南|下：y == t，队列增长方向和 x 轴相反，正南方向（x ＝ 0）数值为 (2t-1)^2 + 3t，所以 v ＝ (2t-1)^2 + 3t - x

西|左：x == -t，队列增长方向和 y 轴相反，正西方向（y ＝ 0）数值为 (2t-1)^2 + 5t，所以 v = (2t-1)^2 + 5t - y

北|上：y == -t，队列增长方向和 x 轴一致，正北方向（x ＝ 0）数值为 (2t-1)^2 + 7t，所以 v ＝ (2t-1)^2 + 7t + x

12. 一个整数，知道位数，如何判断它是否能被3整除，不可以使用除法和模运算

首先 3x=2^n+1时 仅当 n 为奇数才可能 因为2^n = 3x + (-1)^n;所以该问题就转化为了

找到最后一个为1的位a，看看向前的一个1（b）和这个位的距离，如果为偶数的距离则不能整除，如果是奇数，去除b之后的位继续判断

13. seq=[a,b,...,z,aa,ab,...,az,ba,bb...,bz,...za,zb,...,zz,aaa...],求[a-z]+(从a到z任意字符组成的字符串)s在seq的位置，即排在第几

本质就是26进制。

大家都知道，看一个数是否能被2整除只需要看它的个位能否被2整除即可。可是你想过为什么吗？这是因为10能被2整除，因此一个数10a+b能被2整除当且仅当b能被2整除。大家也知道，看一个数能否被3整除只需要看各位数之和是否能被3整除。这又是为什么呢？答案或多或少有些类似：因为10^n-1总能被3整除。2345可以写成2\*(999+1) + 3\*(99+1) + 4\*(9+1) + 5，展开就是2\*999+3\*99+4\*9 + 2+3+4+5。前面带了数字9的项肯定都能被3整除了，于是要看2345能否被3整除就只需要看2+3+4+5能否被3整除了。当然，这种技巧只能在10进制下使用，不过类似的结论可以推广到任意进制。

     注意到36是4的整数倍，而ZZZ...ZZ除以7总是得555...55。也就是说，判断一个36进制数能否被4整除只需要看它的个位，而一个36进制数能被7整除当且仅当各位数之和能被7整除。如果一个数同时能被4和7整除，那么这个数就一定能被28整除。于是问题转化为，有多少个连续句子满足各位数字和是7的倍数，同时最后一个数是4的倍数。这样，我们得到了一个O(n)的算法：用P[i]表示前若干个句子除以7的余数为i有多少种情况，扫描整篇文章并不断更新P数组。当某句话的最后一个字能被4整除时，假设以这句话结尾的前缀和除以7余x，则将此时P[x]的值累加到最后的输出结果中（两个前缀的数字和除以7余数相同，则较长的前缀多出来的部分一定整除7）。

     上述算法是我出这道题的本意，但比赛后我见到了其它各种各样新奇的算法。比如有人注意到36^n mod 28总是等于8，利用这个性质也可以构造出类似的线性算法来。还有人用动态规划（或者说递推）完美地解决了这个问题。我们用f[i,j]表示以句子i结束，除以28余数为j的文本片段有多少个；处理下一句话时我们需要对每一个不同的j进行一次扫描，把f[i-1,j]加进对应的f[i,j']中。最后输出所有的f[i,0]的总和即可。这个动态规划可以用滚动数组，因此它的空间同前面的算法一样也是常数的。

     如果你完全不知道我在说什么，你可以看看和进位制、同余相关的文章。另外，我之前还曾出过一道很类似的题(VOJ1090)，你可以对比着看一看。

有一个整数n,写一个函数f(n),返回0到n之间出现的"1"的个数。比如f(13)=6,现在f(1)=1,问有哪些n能满足f(n)=n？

例如：f(13)=6, 因为1,2,3,4,5,6,7,8,9,10,11,12,13.数数1的个数，正好是6.

public class Test {

public int n = 2;

public int count = 0;

public void BigestNumber(int num) {

for (int i = 1; i <= num; i++) {

int m = 0;

int j = i;

while (j > 0) {

m = j % 10;

if (m == 1)

    count++;

if (j > 0)

    j = j / 10;

}

}

System.out.println("f(" + num + ")=" + count);

}

public static void main(String args[]) {

Test t = new Test();

long begin = System.currentTimeMillis();

t.BigestNumber(10000000);

long end = System.currentTimeMillis();

System.out.println("总时间" + (end-begin)/1000 + "秒");

}

}

结果：

f(10000000)=7000001

总时间5秒

1、将一整数逆序后放入一数组中（要求递归实现）

void convert(int \*result, int n) {

 if(n>=10)

  convert(result+1, n/10);

 \*result = n%10;

}

int main(int argc, char\* argv[]) {

 int n = 123456789, result[20]={};

 convert(result, n);

 printf("%d:", n);

 for(int i=0; i<9; i++)

  printf("%d", result);

}

2、求高于平均分的学生学号及成绩（学号和成绩人工输入）

double find(int total, int n) {

 int number, score,  average;

 scanf("%d", &number);

 if(number != 0) {

  scanf("%d", &score);

  average = find(total+score, n+1);

  if(score >= average)

   printf("%d:%d\n", number, score);

  return average;

 } else {

  printf("Average=%d\n", total/n);

  return total/n;

 }

}

int main(int argc, char\* argv[]) {

 find(0, 0);

}

3、递归实现回文判断（如：abcdedbca就是回文，判断一个面试者对递归理解的简单程序）

int find(char \*str, int n) {

 if(n<=1) return 1;

 else if(str[0]==str[n-1]) return find(str+1, n-2);

 else  return 0;

}

int main(int argc, char\* argv[]) {

 char \*str = "abcdedcba";

 printf("%s: %s\n", str, find(str, strlen(str)) ? "Yes" : "No");

}

4、组合问题（从M个不同字符中任取N个字符的所有组合）

void find(char \*source, char \*result, int n) {

 if(n==1) {

  while(\*source)

     printf("%s%c\n", result, \*source++);

 } else {

  int i, j;

  for(i=0; source != 0; i++);

  for(j=0; result[j] != 0; j++);

  for(; i>=n; i--) {

   result[j] = \*source++;

   result[j+1] = '\0';

   find(source, result, n-1);

  }

 }

}

int main(int argc, char\* argv[]) {

 int const n = 3;

 char \*source = "ABCDE", result[n+1] = {0};

 if(n>0 && strlen(source)>0 && n<=strlen(source))

  find(source, result, 3);

}

5、分解成质因数(如435234=251\*17\*17\*3\*2，据说是华为笔试题)

void prim(int m, int n) {

 if(m>n) {

  while(m%n != 0) n++;

  m /= n;

  prim(m, n);

  printf("%d\*", n);

 }

}

int main(int argc, char\* argv[]) {

 int n = 435234;

 printf("%d=", n);

 prim(n, 2);

}

6、寻找迷宫的一条出路，o：通路； X：障碍。（大家经常谈到的一个小算法题）

#define MAX\_SIZE  8

int H[4] = {0, 1, 0, -1};

int V[4] = {-1, 0, 1, 0};

char Maze[MAX\_SIZE][MAX\_SIZE] = {{'X','X','X','X','X','X','X','X'},

                                 {'o','o','o','o','o','X','X','X'},

                                 {'X','o','X','X','o','o','o','X'},

                             {'X','o','X','X','o','X','X','o'},

                         {'X','o','X','X','X','X','X','X'},

{'X','o','X','X','o','o','o','X'},

         {'X','o','o','o','o','X','o','o'},

                                 {'X','X','X','X','X','X','X','X'}};

void FindPath(int X, int Y) {

    if(X == MAX\_SIZE || Y == MAX\_SIZE) {

       for(int i = 0; i < MAX\_SIZE; i++)

for(int j = 0; j < MAX\_SIZE; j++)

                  printf("%c%c", Maze[j], j < MAX\_SIZE-1 ? ' ' : '\n');

}else for(int k = 0; k < 4; k++)

if(X >= 0 && Y >= 0 && Y < MAX\_SIZE && X < MAX\_SIZE && 'o' == Maze[X][Y]) {

                   Maze[X][Y] = ' ';

                   FindPath(X+V[k], Y+H[k]);

                   Maze[X][Y] ='o';

}

}

int main(int argc, char\* argv[]) {

    FindPath(1,0);

}

7、随机分配座位，共50个学生，使学号相邻的同学座位不能相邻(早些时候用C#写的，没有用C改写）。

static void Main(string[] args)

{

 int Tmp = 0, Count = 50;

 int[] Seats = new int[Count];

 bool[] Students = new bool[Count];

 System.Random RandStudent=new System.Random();

 Students[Seats[0]=RandStudent.Next(0,Count)]=true;

 for(int i = 1; i < Count; ) {

     Tmp=(int)RandStudent.Next(0,Count);

     if((!Students[Tmp])&&(Seats[i-1]-Tmp!=1) && (Seats[i-1] - Tmp) != -1) {

   Seats[i++] = Tmp;

Students[Tmp] = true;

  }

 }

 foreach(int Student in Seats)

     System.Console.Write(Student + " ");

 System.Console.Read();

}

8、求网格中的黑点分布。现有6\*7的网格，在某些格子中有黑点，已知各行与各列中有黑点的点数之和，请在这张网格中画出黑点的位置。（这是一网友提出的题目，说是他笔试时遇到算法题）

#define ROWS 6

#define COLS 7

int iPointsR[ROWS] = {2, 0, 4, 3, 4, 0};           // 各行黑点数和的情况

int iPointsC[COLS] = {4, 1, 2, 2, 1, 2, 1};        // 各列黑点数和的情况

int iCount, iFound;

int iSumR[ROWS], iSumC[COLS], Grid[ROWS][COLS];

int Set(int iRowNo) {

if(iRowNo == ROWS) {

        for(int iColNo=0; iColNo < COLS && iSumC[iColNo]==iPointsC[iColNo]; iColNo++)

           if(iColNo == COLS-1) {

               printf("\nNo.%d:\n", ++iCount);

               for(int i=0; i < ROWS; i++)

                  for(int j=0; j < COLS; j++)

                      printf("%d%c", Grid[j], (j+1) % COLS ? ' ' : '\n');

               iFound = 1; // iFound = 1，有解

           }

    } else {

        for(int iColNo=0; iColNo < COLS; iColNo++) {

            if(iPointsR[iRowNo] == 0) {

                Set(iRowNo + 1);

   } else if(Grid[iRowNo][iColNo]==0) {

Grid[iRowNo][iColNo] = 1;

iSumR[iRowNo]++; iSumC[iColNo]++;                                  if(iSumR[iRowNo]<="iPointsC[iColNo])" <="" div="" style="word-wrap: break-word;">

                     Set(iRowNo);

else if(iSumR[iRowNo]==iPointsR[iRowNo] && iRowNo < ROWS)

                     Set(iRowNo + 1);

                Grid[iRowNo][iColNo] = 0;

                iSumR[iRowNo]--;

iSumC[iColNo]--;

            }

        }

    }

return iFound;     // 用于判断是否有解

}

int main(int argc, char\* argv[]) {

    if(!Set(0))

        printf("Failure!");

}

9、有4种面值的邮票很多枚，这4种邮票面值分别1, 4, 12, 21，现从多张中最多任取5张进行组合，求取出这些邮票的最大连续组合值。（据说是华为2003年校园招聘笔试题）

#define N 5

#define M 5

int k, Found, Flag[N];

int Stamp[M] = {0, 1, 4, 12, 21};

// 在剩余张数n中组合出面值和Value

int Combine(int n, int Value) {

 if(n >= 0 && Value == 0) {

  Found = 1;

  int Sum = 0;

  for(int i=0; i<="" div="" style="word-wrap: break-word;">

   Sum += Stamp[Flag];

   printf("%d ", Stamp[Flag]);

  }

  printf("\tSum=%d\n\n", Sum);

 }else for(int i=1; i0; i++)

  if(Value-Stamp >= 0) {

   Flag[k++] = i;

   Combine(n-1, Value-Stamp);

   Flag[--k] = 0;

  }

 return Found;

}

int main(int argc, char\* argv[]) {

 for(int i=1; Combine(N, i); i++, Found=0);

}

10、大整数数相乘的问题。（这是2002年在一考研班上遇到的算法题）

void Multiple(char A[], char B[], char C[]) {

    int TMP, In=0, LenA=-1, LenB=-1;

    while(A[++LenA] != '\0');

    while(B[++LenB] != '\0');

    int Index, Start = LenA + LenB - 1;

    for(int i=LenB-1; i>=0; i--) {

        Index = Start--;

        if(B != '0') {

            for(int In=0, j=LenA-1; j>=0; j--) {

                TMP = (C[Index]-'0') + (A[j]-'0') \* (B - '0') + In;

                C[Index--] = TMP % 10 + '0';

                In = TMP / 10;

            }

            C[Index] = In + '0';

        }

    }

}

int main(int argc, char\* argv[]) {

    char A[] = "21839244444444448880088888889";

    char B[] = "38888888888899999999999999988";

char C[sizeof(A) + sizeof(B) - 1];

    for(int k=0; k<="" div="" style="word-wrap: break-word;">

        C[k] = '0';

    C[sizeof(C)-1] = '\0';

    Multiple(A, B, C);

    for(int i=0; C != '\0'; i++)

        printf("%c", C);

}

11、求最大连续递增数字串（如“ads3sl456789DF3456ld345AA”中的“456789”）

int GetSubString(char \*strSource, char \*strResult) {

    int iTmp=0, iHead=0, iMax=0;

    for(int Index=0, iLen=0; strSource[Index]; Index++) {

        if(strSource[Index] >= '0' && strSource[Index] <= '9' &&

strSource[Index-1] > '0' && strSource[Index] == strSource[Index-1]+1) {

            iLen++;                       // 连续数字的长度增1

        } else {                          // 出现字符或不连续数字

            if(iLen > iMax) {

            iMax = iLen;  iHead = iTmp;

            }

        // 该字符是数字，但数字不连续

            if(strSource[Index] >= '0' && strSource[Index] <= '9') {

                iTmp = Index;

iLen = 1;

            }

        }

    }

    for(iTmp=0 ; iTmp < iMax; iTmp++) // 将原字符串中最长的连续数字串赋值给结果串

        strResult[iTmp] = strSource[iHead++];

    strResult[iTmp]='\0';

    return iMax;     // 返回连续数字的最大长度

}

int main(int argc, char\* argv[]) {

    char strSource[]="ads3sl456789DF3456ld345AA", char strResult[sizeof(strSource)];

printf("Len=%d, strResult=%s \nstrSource=%s\n",

GetSubString(strSource, strResult), strResult, strSource);

}

12、四个工人，四个任务，每个人做不同的任务需要的时间不同，求任务分配的最优方案。（2005年5月29日全国计算机软件资格水平考试——软件设计师的算法题）。

#include "stdafx.h"

#define N 4

int Cost[N][N] = { {2, 12, 5, 32},  // 行号：任务序号，列号：工人序号

                    {8, 15, 7, 11},  // 每行元素值表示这个任务由不同工人完成所需要的时间

                    {24, 18, 9, 6},

                    {21, 1, 8, 28}};

int MinCost=1000;

int Task[N], TempTask[N], Worker[N];

void Assign(int k, int cost) {

 if(k == N) {

  MinCost = cost;

  for(int i=0; i<="" div="" style="word-wrap: break-word;">

   TempTask = Task;

 } else {

  for(int i=0; i<="" div="" style="word-wrap: break-word;">

   if(Worker==0 && cost+Cost[k] < MinCost) { // 为提高效率而进行剪枝

    Worker = 1; Task[k] = i;

    Assign(k+1, cost+Cost[k]);

    Worker = 0; Task[k] = 0;

   }

  }

 }

}

int main(int argc, char\* argv[]) {

 Assign(0, 0);

 printf("最佳方案总费用=%d\n", MinCost);

 for(int i=0; i<="" div="" style="word-wrap: break-word;">

  printf("\t任务%d由工人%d来做：%d\n", i, TempTask, Cost[TempTask]);

}

13、八皇后问题，输出了所有情况，不过有些结果只是旋转了90度而已。（回溯算法的典型例题，是数据结构书上算法的具体实现，大家都亲自动手写过这个程序吗？）

#define N 8

int Board[N][N];

int Valid(int i, int j) {  // 判断下棋位置是否有效

 int k = 1;

 for(k=1; i>=k && j>=k;k++)

  if(Board[i-k][j-k]) return 0;

 for(k=1; i>=k;k++)

  if(Board[i-k][j])  return 0;

 for(k=1; i>=k && j+k<="" div="" style="word-wrap: break-word;">

  if(Board[i-k][j+k]) return 0;

 return 1;

}

void Trial(int i, int n) {  // 寻找合适下棋位置

 if(i == n) {

  for(int k=0; k<="" div="" style="word-wrap: break-word;">

   for(int m=0; m<="" div="" style="word-wrap: break-word;">

    printf("%d ", Board[k][m]);

   printf("\n");

  }

  printf("\n");

 } else {

  for(int j=0; j<="" div="" style="word-wrap: break-word;">

   Board[j] = 1;

   if(Valid(i,j))

    Trial(i+1, n);

   Board[j] = 0;

  }

 }

}

int main(int argc, char\* argv[]) {

 Trial(0, N);

}

14、实现strstr功能，即在父串中寻找子串首次出现的位置。（笔试中常让面试者实现标准库中的一些函数）

char \* strstring(char \*ParentString, char \*SubString) {

 char \*pSubString, \*pPareString;

 for(char \*pTmp=ParentString; \*pTmp; pTmp++) {

  pSubString = SubString;

  pPareString = pTmp;

  while(\*pSubString == \*pPareString && \*pSubString != '\0') {

   pSubString++;

   pPareString++;

  }

  if(\*pSubString == '\0')  return pTmp;

 }

 return NULL;

}

int main(int argc, char\* argv[]) {

 char \*ParentString = "happy birthday to you!";

 char \*SubString = "birthday";

 printf("%s",strstring(ParentString, SubString));

}

15、现在小明一家过一座桥，过桥的时候是黑夜，所以必须有灯。现在小明过桥要1分，小明的弟弟要3分，小明的爸爸要6分，小明的妈妈要8分，小明的爷爷要12分。每次此桥最多可过两人，而过桥的速度依过桥最慢者而定，而且灯在点燃后30分就会熄灭。问小明一家如何过桥时间最短？（原本是个小小智力题，据说是外企的面试题，在这里用程序来求解）

#include "stdafx.h"

#define N    5

#define SIZE 64

// 将人员编号：小明-0，弟弟-1，爸爸-2，妈妈-3，爷爷-4

// 每个人的当前位置：0--在桥左边， 1--在桥右边

int Position[N];

// 过桥临时方案的数组下标； 临时方案； 最小时间方案；

int Index, TmpScheme[SIZE], Scheme[SIZE];

// 最小过桥时间总和，初始值100；每个人过桥所需要的时间

int MinTime=100, Time[N]={1, 3, 6, 8, 12};

// 寻找最佳过桥方案。Remnant:未过桥人数; CurTime:当前已用时间;

// Direction:过桥方向,1--向右,0--向左

void Find(int Remnant, int CurTime, int Direction) {

    if(Remnant == 0) {                               // 所有人已经过桥，更新最少时间及方案

        MinTime=CurTime;

        for(int i=0; i=0; i++)

            Scheme = TmpScheme;

    } else if(Direction == 1) {                        // 过桥方向向右，从桥左侧选出两人过桥

        for(int i=0; i<="" div="" style="word-wrap: break-word;">

            if(Position == 0 && CurTime + Time < MinTime) {

                TmpScheme[Index++] = i;

                Position = 1;

                for(int j=0; j<="" div="" style="word-wrap: break-word;">

                    int TmpMax = (Time > Time[j] ? Time : Time[j]);

                    if(Position[j] == 0 && CurTime + TmpMax < MinTime) {

                        TmpScheme[Index++] = j;

                        Position[j] = 1;

                        Find(Remnant - 2, CurTime + TmpMax, !Direction);

                        Position[j] = 0;

                        TmpScheme[--Index] = -1;

                    }

                }

                Position = 0;

                TmpScheme[--Index] = -1;

            }

    } else {        // 过桥方向向左，从桥右侧选出一个人回来送灯

        for(int j=0; j<="" div="" style="word-wrap: break-word;">

            if(Position[j] == 1 && CurTime+Time[j] < MinTime) {

                TmpScheme[Index++] = j;

                Position[j] = 0;

                Find(Remnant+1, CurTime+Time[j], !Direction);

                Position[j] = 1;

                TmpScheme[--Index] = -1;

            }

        }

    }

}

int main(int argc, char\* argv[]) {

    for(int i=0; i<="" div="" style="word-wrap: break-word;">

        Scheme = TmpScheme = -1;

Find(N, 0, 1);        // 查找最佳方案

    printf("MinTime=%d:", MinTime); // 输出最佳方案

    for(int i=0; i=0; i+=3)

        printf("  %d-%d  %d", Scheme, Scheme[i+1], Scheme[i+2]);

    printf("\b\b  ");

}

16、2005年11月金山笔试题。编码完成下面的处理函数。函数将字符串中的字符'\*'移到串的前部分，前面的非'\*'字符后移，但不能改变非'\*'字符的先后顺序，函数返回串中字符'\*'的数量。如原始串为：ab\*\*cd\*\*e\*12，处理后为\*\*\*\*\*abcde12，函数并返回值为5。（要求使用尽量少的时间和辅助空间）

int change(char \*str) {

 int count = 0;

 for(int i=0, j=0; str; i++) {

  if(str=='\*') {

   for(j=i-1; str[j]!='\*'&&j>=0; j--)

    str[j+1]=str[j];

   str[j+1] = '\*';

   count++;

  }

 }

 return count;

}

int main(int argc, char\* argv[]) {

 char str[] = "ab\*\*cd\*\*e\*12";

 printf("str1=%s\n", str);

 printf("str2=%s, count=%d", str, change(str));

}

// 终于得到一个比较高效的算法，一个网友提供，估计应该和金山面试官的想法一致。算法如下：

int change(char \*str) {

 int i,j=strlen(str)-1;

 for(i=j; j>=0; j--) {

  if(str!='\*') {

   i--;

  } else if(str[j]!='\*') {

   str = str[j];

   str[j] = '\*';

   i--;

  }

 }

 return i+1;

}

17、2005年11月15日华为软件研发笔试题。实现一单链表的逆转。

#include "stdafx.h"

typedef char eleType;  // 定义链表中的数据类型

typedef struct listnode  { // 定义单链表结构

 eleType data;

 struct listnode \*next;

}node;

node \*create(int n) {  // 创建单链表，n为节点个数

 node \*p = (node \*)malloc(sizeof(node));

 node \*head = p;  head->data = 'A';

 for(int i='B'; i<'A'+n; i++) {

  p = (p->next = (node \*)malloc(sizeof(node)));

  p->data = i;

  p->next = NULL;

 }

 return head;

}

void print(node \*head) { // 按链表顺序输出链表中元素

 for(; head; head = head->next)

  printf("%c ", head->data);

 printf("\n");

}

node \*reverse(node \*head, node \*pre) { // 逆转单链表函数。这是笔试时需要写的最主要函数

 node \*p=head->next;

 head->next = pre;

 if(p) return reverse(p, head);

 else  return head;

}

int main(int argc, char\* argv[]) {

 node \*head = create(6);

 print(head);

 head = reverse(head, NULL);

 print(head);

}

18、编码实现字符串转整型的函数（实现函数atoi的功能），据说是神州数码笔试题。如将字符串 ”+123”?123, ”-0123”?-123, “123CS45”?123, “123.45CS”?123, “CS123.45”?0

#include "stdafx.h"

int str2int(const char \*str) {    // 字符串转整型函数

 int i=0, sign=1, value = 0;

 if(str==NULL)  return NULL;    // 空串直接返回 NULL

 if(str[0]=='-' || str[0]=='+') {   // 判断是否存在符号位

  i = 1;

  sign = (str[0]=='-' ? -1 : 1);

 }

 for(; str>='0' && str<='9'; i++) // 如果是数字，则继续转换

  value = value \* 10 + (str - '0');

 return sign \* value;

}

int main(int argc, char \*argv[]) {

 char \*str = "-123.45CS67";

 int  val  = str2int(str);

 printf("str=%s\tval=%d\n", str, val);

}

19、歌德巴赫猜想。任何一个偶数都可以分解为两个素数之和。（其实这是个C二级考试的模拟试题）

#include "stdafx.h"

#include "math.h"

int main(int argc, char\* argv[]) {

 int Even=78, Prime1, Prime2, Tmp1, Tmp2;

 for(Prime1=3; Prime1<=Even/2; Prime1+=2) {

  for(Tmp1=2,Tmp2=sqrt(float(Prime1)); Tmp1<=Tmp2 && Prime1%Tmp1 != 0; Tmp1++);

  if(Tmp1<=Tmp2) continue;

  Prime2 = Even-Prime1;

  for(Tmp1=2,Tmp2=sqrt(float(Prime2)); Tmp1<=Tmp2 && Prime2%Tmp1 != 0; Tmp1++);

  if(Tmp1<=Tmp2) continue;

  printf("%d=%d+%d\n", Even, Prime1, Prime2);

 }

}

20、快速排序（东软喜欢考类似的算法填空题，又如堆排序的算法等）

#include "stdafx.h"

#define N 10

int part(int list[], int low, int high) {  // 一趟排序，返回分割点位置

 int tmp = list[low];

 while(low<="" div="" style="word-wrap: break-word;">

  while(low=tmp) --high;

  list[low] = list[high];

  while(low<="tmp)"  ++low;="" <="" div="" style="word-wrap: break-word;">

  list[high] = list[low];

 }

 list[low] = tmp;

 return low;

}

void QSort(int list[], int low, int high) { // 应用递归进行快速排序

 if(low<="" div="" style="word-wrap: break-word;">

  int mid = part(list, low, high);

  QSort(list, low, mid-1);

  QSort(list, mid+1, high);

 }

}

void show(int list[], int n) {    // 输出列表中元素

 for(int i=0; i<="" div="" style="word-wrap: break-word;">

  printf("%d ", list);

 printf("\n");

}

int main(int argc, char\* argv[]) {

 int list[N] = {23, 65, 26, 1, 6, 89, 3, 12, 33, 8};

 show(list, N);      // 输出排序前序列

 QSort(list, 0, N-1);     // 快速排序

 show(list, N);      // 输出排序后序列

}

21、2005年11月23日慧通笔试题：写一函数判断某个整数是否为回文数，如12321为回文数。可以用判断入栈和出栈是否相同来实现（略微复杂些），这里是将整数逆序后形成另一整数，判断两个整数是否相等来实现的。

#include "stdafx.h"

int IsEchoNum(int num) {

 int tmp = 0;

 for(int n = num; n; n/=10)

  tmp = tmp \*10 + n%10;

 return tmp==num;

}

int main(int argc, char\* argv[]) {

 int num = 12321;

 printf("%d  %d\n", num, IsEchoNum(num));

}

22、删除字符串中的数字并压缩字符串（神州数码以前笔试题），如字符串”abc123de4fg56”处理后变为”abcdefg”。注意空间和效率。（下面的算法只需要一次遍历，不需要开辟新空间，时间复杂度为O(N)）

#include "stdafx.h"

void delNum(char \*str) {

 int i, j=0;

// 找到串中第一个数字的位子

 for(i=j=0; str && (str<'0' || str>'9'); j=++i);

 // 从串中第一个数字的位置开始，逐个放入后面的非数字字符

 for(; str; i++)

  if(str<'0' || str>'9')

   str[j++] = str;

 str[j] = '\0';

}

int main(int argc, char\* argv[]) {

 char str[] = "abc123ef4g4h5";

 printf("%s\n", str);

 delNum(str);

 printf("%s\n", str);

}

23、求两个串中的第一个最长子串（神州数码以前试题）。如"abractyeyt","dgdsaeactyey"的最大子串为"actyet"。

#include "stdafx.h"

char \*MaxSubString(char \*str1, char \*str2) {

 int i, j, k, index, max=0;

 for(i=0; str1; i++)

  for(j=0; str2[j]; j++) {

   for(k=0; str1[i+k]==str2[j+k] && (str2[i+k] || str1[i+k]); k++);

   if(k>max) {  // 出现大于当前子串长度的子串，则替换子串位置和程度

    index = j; max = k;

   }

  }

 char \*strResult = (char \*)calloc(sizeof(char), max+1);

 for(i=0; i<="" div="" style="word-wrap: break-word;">

  strResult = str2[index++];

 return strResult;

}

int main(int argc, char\* argv[]) {

 char str1[] = "abractyeyt", str2[] = "dgdsaeactyey";

 char \*strResult = MaxSubString(str1, str2);

 printf("str1=%s\nstr2=%s\nMaxSubString=%s\n", str1, str2, strResult);

}

24、不开辟用于交换数据的临时空间，如何完成字符串的逆序(在技术一轮面试中，有些面试官会这样问)

#include "stdafx.h"

void change(char \*str) {

 for(int i=0,j=strlen(str)-1; i<="" div="" style="word-wrap: break-word;">

  str ^= str[j] ^= str ^= str[j];

 }

}

int main(int argc, char\* argv[]) {

 char str[] = "abcdefg";

 printf("strSource=%s\n", str);

 change(str);

 printf("strResult=%s\n", str);

 return getchar();

}

25、删除串中指定的字符（做此题时，千万不要开辟新空间，否则面试官可能认为你不适合做嵌入式开发）

#include "stdafx.h"

void delChar(char \*str, char c) {

 int i, j=0;

 for(i=0; str; i++)

  if(str!=c) str[j++]=str;

 str[j] = '\0';

}

int main(int argc, char\* argv[]) {

 char str[] = "abcdefgh"; // 注意，此处不能写成char \*str = "abcdefgh";

 printf("%s\n", str);

 delChar(str, 'c');

 printf("%s\n", str);

}

26、判断单链表中是否存在环（网上说的笔试题）

#include "stdafx.h"

typedef char eleType;    // 定义链表中的数据类型

typedef struct listnode  {   // 定义单链表结构

 eleType data;

 struct listnode \*next;

}node;

node \*create(int n) {    // 创建单链表，n为节点个数

 node \*p = (node \*)malloc(sizeof(node));

 node \*head = p;  head->data = 'A';

 for(int i='B'; i<'A'+n; i++) {

  p = (p->next = (node \*)malloc(sizeof(node)));

  p->data = i;

  p->next = NULL;

 }

 return head;

}

void addCircle(node \*head, int n) { // 增加环，将链尾指向链中第n个节点

 node \*q, \*p = head;

 for(int i=1; p->next; i++) {

  if(i==n) q = p;

  p = p->next;

 }

 p->next = q;

}

int isCircle(node \*head) {   // 这是笔试时需要写的最主要函数，其他函数可以不写

 node \*p=head,\*q=head;

 while( p->next && q->next) {

  p = p->next;

  if (NULL == (q=q->next->next)) return 0;

  if (p == q) return 1;

 }

 return 0;

}

int main(int argc, char\* argv[]) {

 node \*head = create(12);

 addCircle(head, 8);   // 注释掉此行，连表就没有环了

 printf("%d\n", isCircle(head));

}