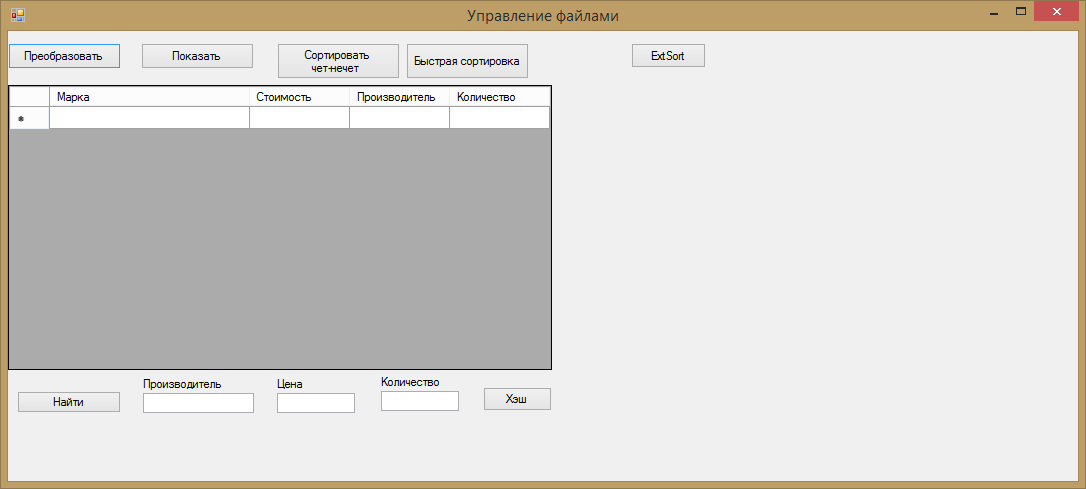
Отчет по программе FileMngm

**Модель:** При переводе из .txt файла в .dat мы использовали типизизирование; при сортировке я применял параллельный вариант сортировки чет –нечет; при поиске использовал дихотомический поиск; также выполнял хеширование файла; в сортировке, поиске и хешировании пользовался методом создания ключей.

**Описание интерфейса:**



Кнопка «Преобразовать» файл типа .txt (не типизированный) преобразует в типизированный файл типа .dat; кнопка «Показать» выводит в DataGridView преобразованный .dat файл; кнопки «Сортировать чет-нечет» и «Быстрая сортировка» выполняют сортировку; кнопка «Найти» выпоняет поиск по введенным полям; кнопка «Хэш» выполняет хэширование и поиск по введенным полям; кнопка «ExtSort» выполняе внешнюю сортировку.

**Описание программы:**

FileMngmC

public unsafe struct TbPhone

{

public fixed byte Maker[25];

public double cost;

public int count;

public fixed byte Mrka[40];

}

public unsafe void Convert (string \_fn,DataGridView \_dgErr)

{

TbPhone[] bPhoneMassiv = new TbPhone[4];

int i, j, Res;

string tmp;

TbPhone\* bPhone = stackalloc TbPhone[1];

byte[] bbPhone = new byte[sizeof(TbPhone)];

string bPhoneTXT;

string[] rPhone;

StreamReader fPhonTXT = new StreamReader(\_fn);

tmp = "";

for (i = 0; i <= \_fn.IndexOf("."); i++)

tmp = tmp + \_fn[i];

tmp = tmp + "dat";

FileStream fPhon = File.OpenWrite(tmp);

j = 0;

while ((bPhoneTXT = fPhonTXT.ReadLine()) != null)

{

Res = 0;

rPhone = bPhoneTXT.Split(new Char[] { ';' });

if (rPhone.Length == 4)

{

for (i = 0; i < 25; i++)

bPhone->Maker[i] = 32;

for (i = 0; i < rPhone[0].Length; i++)

bPhone->Maker[i] = System.Convert.ToByte(rPhone[0][i]);

for (i = 0; i < 40; i++)

bPhone->Mrka[i] = 32;

for (i = 0; i < rPhone[3].Length; i++)

bPhone->Mrka[i] = System.Convert.ToByte(rPhone[3][i]);

if (Double.TryParse(rPhone[1], out bPhone->cost))

{

if (Int32.TryParse(rPhone[2], out bPhone->count))

Res = 0;

else

Res = 2;

}

else

Res = 1;

}

else

Res = 3;

if (Res != 0)

{

\_dgErr.Visible = true;

\_dgErr.RowCount = \_dgErr.RowCount + 1;

\_dgErr.Rows[j].Cells[0].Value = Res.ToString() + "-" + bPhoneTXT;

j = j + 1;

}

else

{

Marshal.Copy((IntPtr)bPhone, bbPhone, 0, bbPhone.Length);

fPhon.Write(bbPhone, 0, bbPhone.Length);

}

}

fPhon.Close();

}

public unsafe void Load (string \_fn, DataGridView \_dgPhone)

{

int i;

int flen;

string s1;

string s2;

FileMngmC.TbPhone\* bPhone = stackalloc FileMngmC.TbPhone[1];

byte[] bbPhone = new byte[sizeof(FileMngmC.TbPhone)];

FileStream fPhone = File.OpenRead(\_fn);

flen = (Int32)fPhone.Length / sizeof(FileMngmC.TbPhone);

\_dgPhone.RowCount = flen + 1;

for (i = 0; i < flen; i++)

{

fPhone.Read(bbPhone, 0, bbPhone.Length);

Marshal.Copy(bbPhone, 0, (IntPtr)bPhone, bbPhone.Length);

s1 = Encoding.UTF8.GetString(bbPhone, 0, 25);

s2 = Encoding.UTF8.GetString(bbPhone, 44, 40);

\_dgPhone.Rows[i].Cells[2].Value = s1;

\_dgPhone.Rows[i].Cells[1].Value = bPhone->cost.ToString();

\_dgPhone.Rows[i].Cells[3].Value = bPhone->count.ToString();

\_dgPhone.Rows[i].Cells[0].Value = s2;

}

fPhone.Close();

}

FileMngmSDS

public unsafe void FileSort(string \_fn)

{

int i, j, k,jm,jM,km,kM;

int flen;

FileMngmC.TbPhone\* bPhone = stackalloc FileMngmC.TbPhone[1];

byte[] bbPhone = new byte[sizeof(FileMngmC.TbPhone)];

FileStream fPhone = File.OpenRead(\_fn);

flen = (Int32)fPhone.Length / sizeof(FileMngmC.TbPhone);

FileMngmC.TbPhone[] MinPhone = new FileMngmC.TbPhone[flen];

FileMngmC.TbPhone[] MaxPhone = new FileMngmC.TbPhone[flen];

FileMngmC.TbPhone[] mMinPhone = new FileMngmC.TbPhone[flen];

FileMngmC.TbPhone[] MMinPhone = new FileMngmC.TbPhone[flen];

FileMngmC.TbPhone[] mMaxPhone = new FileMngmC.TbPhone[flen];

FileMngmC.TbPhone[] MMaxPhone = new FileMngmC.TbPhone[flen];

FileMngmC.TbPhone[] arPhone = new FileMngmC.TbPhone[flen];

FileMngmC.TbPhone opPhone;

for (i = 0; i < flen; i++)

{

fPhone.Read(bbPhone, 0, bbPhone.Length);

Marshal.Copy(bbPhone, 0, (IntPtr)bPhone, bbPhone.Length);

arPhone[i] = bPhone[0];

}

opPhone = arPhone[flen / 2];

k = 0;

j = 0;

for (i = 0; i < flen; i++)

if (String.Compare(Key\_ar(i, arPhone), Key\_tmp(opPhone)) <= 0)

{

MinPhone[j] = arPhone[i];

j = j + 1;

}

else

{

MaxPhone[k] = arPhone[i];

k = k + 1;

}

opPhone = MinPhone[j / 2];

jm = 0;

jM = 0;

for (i = 0; i < j; i++)

if (String.Compare(Key\_ar(i, MinPhone), Key\_tmp(opPhone)) <= 0)

{

mMinPhone[jm] = MinPhone[i];

jm = jm + 1;

}

else

{

MMinPhone[jM] = MinPhone[i];

jM = jM + 1;

}

opPhone = MaxPhone[k / 2];

km = 0;

kM = 0;

for (i = 0; i < k; i++)

if (String.Compare(Key\_ar(i, MaxPhone), Key\_tmp(opPhone)) <= 0)

{

mMaxPhone[km] = MaxPhone[i];

km = km + 1;

}

else

{

MMaxPhone[kM] = MaxPhone[i];

kM = kM + 1;

}

Thread thrmMin = new Thread(() => Sort1(mMinPhone));

Thread thrmMax = new Thread(() => Sort2(mMaxPhone));

Thread thrMMin = new Thread(() => Sort3(MMinPhone));

Thread thrMMax = new Thread(() => Sort4(MMaxPhone));

thrmMin.Priority = ThreadPriority.Highest;

thrmMax.Priority = ThreadPriority.Highest;

thrMMin.Priority = ThreadPriority.Highest;

thrMMin.Priority = ThreadPriority.Highest;

thrmMin.Start();

thrMMin.Start();

thrmMax.Start();

thrMMax.Start();

while (thrmMin.IsAlive | thrMMin.IsAlive | thrmMax.IsAlive | thrMMax.IsAlive);

for (i = 0; i < jm; i++)

arPhone[i] = mMinPhone[(flen - jm) + i];

for (i = 0; i < jM; i++)

arPhone[jm + i] = MMinPhone[(flen - jM) + i];

for (i = 0; i < km; i++)

arPhone[jm + jM + i] = mMaxPhone[(flen - km) + i];

for (i = 0; i < kM; i++)

arPhone[jm + jM + km + i] = MMaxPhone[(flen - kM) + i];

fPhone.Close();

fPhone = File.OpenWrite(\_fn);

for (i = 0; i < flen; i++)

{

bPhone[0] = arPhone[i];

Marshal.Copy((IntPtr)bPhone, bbPhone, 0, bbPhone.Length);

fPhone.Write(bbPhone, 0, bbPhone.Length);

}

fPhone.Close();

}

private unsafe string Key\_ar(int \_k,FileMngmC.TbPhone[] \_arPhone)

{

string key\_ar;

int k;

FileMngmC.TbPhone\* bPhone = stackalloc FileMngmC.TbPhone[1];

key\_ar = \_arPhone[\_k].count.ToString("D3");

bPhone[0] = \_arPhone[\_k];

for (k = 0; k < 25; k++)

key\_ar =key\_ar + bPhone->Maker[k].ToString();

key\_ar = key\_ar + \_arPhone[\_k].cost.ToString("0.0");

return key\_ar;

}

private unsafe string Key\_tmp(FileMngmC.TbPhone \_tmpPhone)

{

string key;

int k;

key = \_tmpPhone.count.ToString("D3");

for (k = 0; k < 25; k++)

key = key + \_tmpPhone.Maker[k].ToString();

key = key + \_tmpPhone.cost.ToString("0.0");

return key;

}

public void Sort1 (FileMngmC.TbPhone[] \_arPhone)

{

int i, j;

bool eoc;

string key\_tmp, key\_ar;

FileMngmC.TbPhone tmpPhone;

for (i = 1; i < \_arPhone.Length; i++)

{

tmpPhone = \_arPhone[i];

key\_tmp=Key\_tmp(tmpPhone);

j = i - 1;

eoc = false;

while (!eoc)

{

key\_ar = Key\_ar(j, \_arPhone);

if (String.Compare(key\_ar, key\_tmp) <= 0)

eoc = true;

else

{

\_arPhone[j + 1] = \_arPhone[j];

j = j - 1;

if (j < 0)

eoc = true;

}

}

\_arPhone[j + 1] = tmpPhone;

}

}

public void Sort2(FileMngmC.TbPhone[] \_arPhone)

{

int i, j;

bool eoc;

string key\_tmp, key\_ar;

FileMngmC.TbPhone tmpPhone;

for (i = 1; i < \_arPhone.Length; i++)

{

tmpPhone = \_arPhone[i];

key\_tmp = Key\_tmp(tmpPhone);

j = i - 1;

eoc = false;

while (!eoc)

{

key\_ar = Key\_ar(j, \_arPhone);

if (String.Compare(key\_ar, key\_tmp) <= 0)

eoc = true;

else

{

\_arPhone[j + 1] = \_arPhone[j];

j = j - 1;

if (j < 0)

eoc = true;

}

}

\_arPhone[j + 1] = tmpPhone;

}

}

public void Sort3(FileMngmC.TbPhone[] \_arPhone)

{

int i, j;

bool eoc;

string key\_tmp, key\_ar;

FileMngmC.TbPhone tmpPhone;

for (i = 1; i < \_arPhone.Length; i++)

{

tmpPhone = \_arPhone[i];

key\_tmp = Key\_tmp(tmpPhone);

j = i - 1;

eoc = false;

while (!eoc)

{

key\_ar = Key\_ar(j, \_arPhone);

if (String.Compare(key\_ar, key\_tmp) <= 0)

eoc = true;

else

{

\_arPhone[j + 1] = \_arPhone[j];

j = j - 1;

if (j < 0)

eoc = true;

}

}

\_arPhone[j + 1] = tmpPhone;

}

}

public void Sort4(FileMngmC.TbPhone[] \_arPhone)

{

int i, j;

bool eoc;

string key\_tmp, key\_ar;

FileMngmC.TbPhone tmpPhone;

for (i = 1; i < \_arPhone.Length; i++)

{

tmpPhone = \_arPhone[i];

key\_tmp = Key\_tmp(tmpPhone);

j = i - 1;

eoc = false;

while (!eoc)

{

key\_ar = Key\_ar(j, \_arPhone);

if (String.Compare(key\_ar, key\_tmp) <= 0)

eoc = true;

else

{

\_arPhone[j + 1] = \_arPhone[j];

j = j - 1;

if (j < 0)

eoc = true;

}

}

\_arPhone[j + 1] = tmpPhone;

}

}

FileMngmDS

public int \_count;

public double \_cost;

public string \_Maker;

public unsafe int DSearch(string \_fn)

{

int i;

int right, middle, left;

int flen;

string key;

string SI;

FileMngmC.TbPhone\* bPhone = stackalloc FileMngmC.TbPhone[1];

byte[] bbPhone = new byte[sizeof(FileMngmC.TbPhone)];

FileStream fPhone = File.OpenRead(\_fn);

flen = (Int32)fPhone.Length / sizeof(FileMngmC.TbPhone);

left = 0;

right = flen - 1;

key = "";

SI = \_count.ToString("D3") + \_Maker + \_cost.ToString("0.0");

while (left < right)

{

middle = (left + right) / 2;

fPhone.Seek(middle \* sizeof(FileMngmC.TbPhone), SeekOrigin.Begin);

fPhone.Read(bbPhone, 0, bbPhone.Length);

Marshal.Copy(bbPhone, 0, (IntPtr)bPhone, bbPhone.Length);

key = bPhone->count.ToString("D3");

for (i = 0; i < 25; i++)

key = key + Convert.ToChar(bPhone->Maker[i]);

key = key + bPhone->cost.ToString("0.0");

if (String.Compare(key, SI, true) < 0)

left = middle + 1;

else

right = middle;

}

fPhone.Seek(right \* sizeof(FileMngmC.TbPhone), SeekOrigin.Begin);

fPhone.Read(bbPhone, 0, bbPhone.Length);

Marshal.Copy(bbPhone, 0, (IntPtr)bPhone, bbPhone.Length);

key =bPhone->count.ToString("D3");

for (i = 0; i < 25; i++)

key = key + Convert.ToChar(bPhone->Maker[i]);

key = key + bPhone->cost.ToString("0.0");

if (String.Compare(key, SI, true) <= 0)

return right;

else

return -1;

}

public bool Put (string \_tbCount, string \_tbCost, string \_tbMaker)

{

int i;

bool tmp;

tmp = true;

tmp = Int32.TryParse(\_tbCount, out \_count) & Double.TryParse(\_tbCost, out \_cost);

\_Maker = \_tbMaker;

for (i = 0; i < (25 - \_tbMaker.Length); i++)

\_Maker =\_Maker+ " ";

return tmp;

}

FileMngmES

string[] wfn = new string[4] { "f1.wrk", "f2.wrk", "f3.wrk", "f4.wrk" };

int fswitch, fgswitch;

static int psize = 3;

static FileMngmC.TbPhone[] arPhone = new FileMngmC.TbPhone[psize];

FileStream[] wf = new FileStream[4];

FileMngmC.TbPhone tmpPhone;

public void ExSort(string \_fn)

{

Divide(\_fn);

}

public unsafe void Divide(string \_fn)

{

int i, j, k, lena;

int flen;

FileMngmC.TbPhone\* bPhone = stackalloc FileMngmC.TbPhone[1];

byte[] bbPhone = new byte[sizeof(FileMngmC.TbPhone)];

File.Delete(wfn[0]);

File.Delete(wfn[1]);

wf[0] = File.OpenWrite(wfn[0]);

wf[1] = File.OpenWrite(wfn[1]);

FileStream fPhone = File.OpenRead(\_fn);

flen = (Int32)fPhone.Length / sizeof(FileMngmC.TbPhone);

fswitch = 0;

lena = 0;

for (i = 0; i < flen; i++)

{

if ((i > (psize - 1)) & (i % psize == 0))

{

InsSort(lena);

for (k = 0; k < lena; k++)

{

bPhone[0] = arPhone[k];

Marshal.Copy((IntPtr)bPhone, bbPhone, 0, bbPhone.Length);

wf[fswitch].Write(bbPhone, 0, bbPhone.Length);

}

fswitch = (fswitch + 1) % 2;

fPhone.Read(bbPhone, 0, bbPhone.Length);

Marshal.Copy(bbPhone, 0, (IntPtr)bPhone, bbPhone.Length);

arPhone[i % psize] = bPhone[0];

lena = 1;

}

else

{

fPhone.Read(bbPhone, 0, bbPhone.Length);

Marshal.Copy(bbPhone, 0, (IntPtr)bPhone, bbPhone.Length);

arPhone[i % psize] = bPhone[0];

lena = lena + 1;

}

}

InsSort(lena);

for (k = 0; k < lena; k++)

{

bPhone[0] = arPhone[k];

Marshal.Copy((IntPtr)bPhone, bbPhone, 0, bbPhone.Length);

wf[fswitch].Write(bbPhone, 0, bbPhone.Length);

}

fPhone.Close();

wf[0].Close();

wf[1].Close();

}

private void InsSort(int \_lena)

{

int k, j;

string key\_ar, key\_tmp;

bool eoc;

for (k = 1; k < \_lena; k++)

{

tmpPhone = arPhone[k];

key\_tmp = tmpPhone.count.ToString("D3") + tmpPhone.cost.ToString("0.0");

j = k - 1;

eoc = false;

while (!eoc)

{

key\_ar = arPhone[j].count.ToString("D3") + arPhone[j].cost.ToString("0.0");

if (String.Compare(key\_ar, key\_tmp) <= 0)

eoc = true;

else

{

arPhone[j + 1] = arPhone[j];

j = j - 1;

if (j < 0)

eoc = true;

}

}

arPhone[j + 1] = tmpPhone;

}

}

public unsafe string Merge()

{

int[] flen = new int[2];

FileMngmC.TbPhone\* bw0 = stackalloc FileMngmC.TbPhone[1];

byte[] bbw0 = new byte[sizeof(FileMngmC.TbPhone)];

FileMngmC.TbPhone\* bw1 = stackalloc FileMngmC.TbPhone[1];

byte[] bbw1 = new byte[sizeof(FileMngmC.TbPhone)];

int wpsize = psize;

bool eopm, eom;

string keyw0, keyw1, fres;

int ip0, ip1;

eom = false;

fgswitch = 0;

fres = "";

while (!eom)

{

eopm = false;

fswitch = 0;

wf[0] = File.OpenRead(wfn[fgswitch]);

flen[0] = (Int32)wf[0].Length / sizeof(FileMngmC.TbPhone);

wf[1] = File.OpenRead(wfn[fgswitch + 1]);

flen[1] = (Int32)wf[1].Length / sizeof(FileMngmC.TbPhone);

File.Delete(wfn[(fgswitch + 2) % 4]);

File.Delete(wfn[(fgswitch + 2) % 4 + 1]);

wf[2] = File.OpenWrite(wfn[(fgswitch + 2) % 4]);

wf[3] = File.OpenWrite(wfn[(fgswitch + 2) % 4 + 1]);

keyw0 = "";

keyw1 = "";

wf[0].Read(bbw0, 0, bbw0.Length);

Marshal.Copy(bbw0, 0, (IntPtr)bw0, bbw0.Length);

keyw0 = bw0->count.ToString("D3") + bw0->cost.ToString("0.0");

flen[0] = flen[0] - 1;

ip0 = 1;

wf[1].Read(bbw1, 0, bbw1.Length);

Marshal.Copy(bbw1, 0, (IntPtr)bw1, bbw1.Length);

keyw1 = bw1->count.ToString("D3") + bw1->cost.ToString("0.0");

flen[1] = flen[1] - 1;

ip1 = 1;

while (!eopm)

{

switch (keyw0.CompareTo(keyw1))

{

case -1:

Marshal.Copy((IntPtr)bw0, bbw0, 0, bbw0.Length);

wf[2 + fswitch].Write(bbw0, 0, bbw0.Length);

if ((ip0 < wpsize) & (flen[0] > 0))

{

wf[0].Read(bbw0, 0, bbw0.Length);

Marshal.Copy(bbw0, 0, (IntPtr)bw0, bbw0.Length);

keyw0 = bw0->count.ToString("D3") + bw0->cost.ToString("0.0");

ip0 = ip0 + 1;

flen[0] = flen[0] - 1;

}

else

keyw0 = "9999";

break;

case 0:

Marshal.Copy((IntPtr)bw0, bbw0, 0, bbw0.Length);

wf[2 + fswitch].Write(bbw0, 0, bbw0.Length);

if ((ip0 < wpsize) & (flen[0] > 0))

{

wf[0].Read(bbw0, 0, bbw0.Length);

Marshal.Copy(bbw0, 0, (IntPtr)bw0, bbw0.Length);

keyw0 = bw0->count.ToString("D3") + bw0->cost.ToString("0.0");

ip0 = ip0 + 1;

flen[0] = flen[0] - 1;

}

else

keyw0 = "9999";

Marshal.Copy((IntPtr)bw1, bbw1, 0, bbw1.Length);

wf[2 + fswitch].Write(bbw1, 0, bbw1.Length);

if ((ip1 < wpsize) & (flen[1] > 0))

{

wf[1].Read(bbw1, 0, bbw1.Length);

Marshal.Copy(bbw1, 0, (IntPtr)bw1, bbw1.Length);

keyw1 = bw1->count.ToString("D3") + bw1->cost.ToString("0.0");

ip1 = ip1 + 1;

flen[1] = flen[1] - 1;

}

else

keyw1 = "9999";

break;

case 1:

Marshal.Copy((IntPtr)bw1, bbw1, 0, bbw1.Length);

wf[2 + fswitch].Write(bbw1, 0, bbw1.Length);

if ((ip1 < wpsize) & (flen[1] > 0))

{

wf[1].Read(bbw1, 0, bbw1.Length);

Marshal.Copy(bbw1, 0, (IntPtr)bw1, bbw1.Length);

keyw1 = bw1->count.ToString("D3") + bw1->cost.ToString("0.0");

ip1 = ip1 + 1;

flen[1] = flen[1] - 1;

}

else

keyw1 = "9999";

break;

} //switch

if ((keyw0.CompareTo("9999") == 0) & (keyw1.CompareTo("9999") == 0))

{

if ((flen[0] == 0) & (flen[1] == 0))

eopm = true;

else

if (!(flen[0] == 0))

{

keyw0 = "";

wf[0].Read(bbw0, 0, bbw0.Length);

Marshal.Copy(bbw0, 0, (IntPtr)bw0, bbw0.Length);

keyw0 = bw0->count.ToString("D3") + bw0->cost.ToString("0.0");

flen[0] = flen[0] - 1;

ip0 = 1;

}

if (!(flen[1] == 0))

{

keyw1 = "";

wf[1].Read(bbw1, 0, bbw1.Length);

Marshal.Copy(bbw1, 0, (IntPtr)bw1, bbw1.Length);

keyw1 = bw1->count.ToString("D3") + bw1->cost.ToString("0.0");

flen[1] = flen[1] - 1;

ip1 = 1;

}

fswitch = (fswitch + 1) % 2;

}

} // eopm

if ((wf[2].Length > 0) & (wf[3].Length > 0))

{

fgswitch = (fgswitch + 2) % 4;

wpsize = wpsize \* 2;

}

else

{

if (wf[2].Length > 0)

fres = wfn[(fgswitch + 2) % 4];

else

fres = wfn[(fgswitch + 2) % 4 + 1];

eom = true;

}

wf[0].Close();

wf[1].Close();

wf[2].Close();

wf[3].Close();

} // eom

return fres;

}

FileMngmHSH

int b = 2;

int h = 505;

public int \_count;

public double \_cost;

public string \_Maker;

public int Hash(string \_key)

{

long e, code;

int i;

e = 1;

code = 0;

for (i = 0; i < \_key.Length; i++)

{

code = code + Convert.ToInt32(\_key[i])\*e;

e = e \* b;

}

return(int)(code % h);

}

public unsafe void CreHSH(string \_fn, DataGridView \_dgHCodes)

{

int i, j, k;

int flen;

int hcode;

string tmp, fnh;

FileMngmC.TbPhone\* bPhone = stackalloc FileMngmC.TbPhone[1];

byte[] bbPhone = new byte[sizeof(FileMngmC.TbPhone)];

FileMngmC.TbPhone tmpPhone;

fnh = "";

for (i = 0; i <= \_fn.IndexOf("."); i++)

fnh = fnh + \_fn[i];

fnh = fnh + "hsh";

FileStream fPhoneH = File.OpenWrite(fnh);

tmpPhone.count = 0;

tmpPhone.cost = 0;

for (i = 0; i < 25; i++)

bPhone->Maker[i] = 32;

flen = h;

\_dgHCodes.Visible = true;

\_dgHCodes.RowCount = 1;

j = 0;

for (i = 0; i < flen; i++)

{

Marshal.Copy((IntPtr)bPhone, bbPhone, 0, bbPhone.Length);

fPhoneH.Write(bbPhone, 0, bbPhone.Length);

}

FileStream fPhone = File.OpenRead(\_fn);

flen = (Int32)fPhone.Length / sizeof(FileMngmC.TbPhone);

for (i = 0; i < flen; i++)

{

fPhone.Read(bbPhone, 0, bbPhone.Length);

Marshal.Copy(bbPhone, 0, (IntPtr)bPhone, bbPhone.Length);

tmpPhone = bPhone[0];

tmp = tmpPhone.count.ToString("D3");

for (k = 0; k < 25; k++)

tmp = tmp + Convert.ToChar(tmpPhone.Maker[k]);

tmp = tmp + tmpPhone.cost.ToString("0.0");

hcode = Hash(tmp);

fPhoneH.Seek(hcode \* sizeof(FileMngmC.TbPhone), SeekOrigin.Begin);

fPhoneH.Write(bbPhone, 0, bbPhone.Length);

\_dgHCodes.RowCount = \_dgHCodes.RowCount + 1;

\_dgHCodes.Rows[j].Cells[0].Value = hcode.ToString("D2") + " - " + tmp;

j = j + 1;

}

fPhone.Close();

fPhoneH.Close();

}

public unsafe string HSearch(string \_fn)

{

int hcode, k;

string tmp , \_SI;

FileMngmC.TbPhone\* bPhone = stackalloc FileMngmC.TbPhone[1];

byte[] bbPhone = new byte[sizeof(FileMngmC.TbPhone)];

FileStream fPhoneH = File.OpenRead(\_fn);

\_SI= \_count.ToString("D3") + \_Maker + \_cost.ToString("0.0");

hcode = Hash(\_SI);

fPhoneH.Seek(hcode \* sizeof(FileMngmC.TbPhone), SeekOrigin.Begin);

fPhoneH.Read(bbPhone, 0, bbPhone.Length);

Marshal.Copy(bbPhone, 0, (IntPtr)bPhone, bbPhone.Length);

tmp = bPhone->count.ToString("D3");

for (k = 0; k < 25; k++)

tmp = tmp + Convert.ToChar(bPhone->Maker[k]);

tmp = tmp + bPhone->cost.ToString("0.0");

if (String.Compare(tmp, \_SI) == 0)

return "YES -" + hcode.ToString();

else

return "No";

fPhoneH.Close();

}

public bool Put(string \_tbCount, string \_tbCost, string \_tbMaker)

{

int i;

bool tmp;

tmp = true;

tmp = Int32.TryParse(\_tbCount, out \_count) & Double.TryParse(\_tbCost, out \_cost);

\_Maker = \_tbMaker;

for (i = 0; i < (25 - \_tbMaker.Length); i++)

\_Maker = \_Maker + " ";

return tmp;

}

FileMngmSP

public FileMngmC.TbPhone[] arPhone;

public unsafe void SortPar (string \_fn)

{

int i;

int flen;

FileMngmC.TbPhone\* bPhone = stackalloc FileMngmC.TbPhone[1];

byte[] bbPhone = new byte[sizeof(FileMngmC.TbPhone)];

FileStream fPhone = File.OpenRead(\_fn);

flen = (Int32)fPhone.Length / sizeof(FileMngmC.TbPhone);

arPhone = new FileMngmC.TbPhone[flen];

for (i = 0; i < flen; i++)

{

fPhone.Read(bbPhone, 0, bbPhone.Length);

Marshal.Copy(bbPhone, 0, (IntPtr)bPhone, bbPhone.Length);

arPhone[i] = bPhone[0];

}

for (i = 0; i< 4; i++)

if ((i % 2) == 0)

{

Thread thrS1 = new Thread(Sort1);

Thread thrS2 = new Thread(Sort2);

Thread thrS3 = new Thread(Sort3);

Thread thrS4 = new Thread(Sort4);

thrS1.Priority = ThreadPriority.Highest;

thrS2.Priority = ThreadPriority.Highest;

thrS3.Priority = ThreadPriority.Highest;

thrS4.Priority = ThreadPriority.Highest;

thrS1.Start();

thrS2.Start();

thrS3.Start();

thrS4.Start();

while (thrS1.IsAlive | thrS2.IsAlive | thrS3.IsAlive | thrS4.IsAlive);

}

else

{

Thread thrS5 = new Thread(Sort5);

Thread thrS6 = new Thread(Sort6);

Thread thrS7 = new Thread(Sort7);

thrS5.Priority = ThreadPriority.Highest;

thrS6.Priority = ThreadPriority.Highest;

thrS7.Priority = ThreadPriority.Highest;

thrS5.Start();

thrS6.Start();

thrS7.Start();

while (thrS5.IsAlive | thrS6.IsAlive | thrS7.IsAlive) ;

}

fPhone.Close();

fPhone = File.OpenWrite(\_fn);

for (i = 0; i < flen; i++)

{

bPhone[0] = arPhone[i];

Marshal.Copy((IntPtr)bPhone, bbPhone, 0, bbPhone.Length);

fPhone.Write(bbPhone, 0, bbPhone.Length);

}

fPhone.Close();

}

private unsafe string Key\_ar(int \_k, FileMngmC.TbPhone[] \_arPhone)

{

string key\_ar;

int k;

FileMngmC.TbPhone\* bPhone = stackalloc FileMngmC.TbPhone[1];

key\_ar = \_arPhone[\_k].count.ToString("D3");

bPhone[0] = \_arPhone[\_k];

for (k = 0; k < 3; k++)

key\_ar = key\_ar + bPhone->Maker[k].ToString();

key\_ar = key\_ar + \_arPhone[\_k].cost.ToString("0.0");

return key\_ar;

}

private unsafe string Key\_tmp(FileMngmC.TbPhone \_tmpPhone)

{

string key;

int k;

key = \_tmpPhone.count.ToString("D3");

for (k = 0; k < 3; k++)

key = key + \_tmpPhone.Maker[k].ToString();

key = key + \_tmpPhone.cost.ToString("0.0");

return key;

}

public void Sort1()

{

int i, j;

int flen = arPhone.Length;

bool eoc;

string key\_tmp, key\_ar;

FileMngmC.TbPhone tmpPhone;

for (i = 1; i < (flen/4); i++)

{

tmpPhone = arPhone[i];

key\_tmp = Key\_tmp(tmpPhone);

j = i - 1;

eoc = false;

while (!eoc)

{

key\_ar = Key\_ar(j, arPhone);

if (String.Compare(key\_ar, key\_tmp) <= 0)

eoc = true;

else

{

arPhone[j + 1] = arPhone[j];

j = j - 1;

if (j < 0)

eoc = true;

}

}

arPhone[j + 1] = tmpPhone;

}

}

public void Sort2()

{

int i, j;

int flen = arPhone.Length;

bool eoc;

string key\_tmp, key\_ar;

FileMngmC.TbPhone tmpPhone;

for (i = (flen/4)+1; i < (flen / 2); i++)

{

tmpPhone = arPhone[i];

key\_tmp = Key\_tmp(tmpPhone);

j = i - 1;

eoc = false;

while (!eoc)

{

key\_ar = Key\_ar(j, arPhone);

if (String.Compare(key\_ar, key\_tmp) <= 0)

eoc = true;

else

{

arPhone[j + 1] = arPhone[j];

j = j - 1;

if (j < 0)

eoc = true;

}

}

arPhone[j + 1] = tmpPhone;

}

}

public void Sort3()

{

int i, j;

int flen = arPhone.Length;

bool eoc;

string key\_tmp, key\_ar;

FileMngmC.TbPhone tmpPhone;

for (i = (flen / 2)+1; i < (3\*flen / 4); i++)

{

tmpPhone = arPhone[i];

key\_tmp = Key\_tmp(tmpPhone);

j = i - 1;

eoc = false;

while (!eoc)

{

key\_ar = Key\_ar(j, arPhone);

if (String.Compare(key\_ar, key\_tmp) <= 0)

eoc = true;

else

{

arPhone[j + 1] = arPhone[j];

j = j - 1;

if (j < 0)

eoc = true;

}

}

arPhone[j + 1] = tmpPhone;

}

}

public void Sort4()

{

int i, j;

int flen = arPhone.Length;

bool eoc;

string key\_tmp, key\_ar;

FileMngmC.TbPhone tmpPhone;

for (i = (3\*flen / 4)+1; i < flen; i++)

{

tmpPhone = arPhone[i];

key\_tmp = Key\_tmp(tmpPhone);

j = i - 1;

eoc = false;

while (!eoc)

{

key\_ar = Key\_ar(j, arPhone);

if (String.Compare(key\_ar, key\_tmp) <= 0)

eoc = true;

else

{

arPhone[j + 1] = arPhone[j];

j = j - 1;

if (j < 0)

eoc = true;

}

}

arPhone[j + 1] = tmpPhone;

}

}

public void Sort5()

{

int i, j;

int flen = arPhone.Length;

bool eoc;

string key\_tmp, key\_ar;

FileMngmC.TbPhone tmpPhone;

for (i = (flen / 8)+1; i < (3\*flen / 8); i++)

{

tmpPhone = arPhone[i];

key\_tmp = Key\_tmp(tmpPhone);

j = i - 1;

eoc = false;

while (!eoc)

{

key\_ar = Key\_ar(j, arPhone);

if (String.Compare(key\_ar, key\_tmp) <= 0)

eoc = true;

else

{

arPhone[j + 1] = arPhone[j];

j = j - 1;

if (j < 0)

eoc = true;

}

}

arPhone[j + 1] = tmpPhone;

}

}

public void Sort6()

{

int i, j;

int flen = arPhone.Length;

bool eoc;

string key\_tmp, key\_ar;

FileMngmC.TbPhone tmpPhone;

for (i = (3\*flen / 8)+1; i < (5\*flen / 8); i++)

{

tmpPhone = arPhone[i];

key\_tmp = Key\_tmp(tmpPhone);

j = i - 1;

eoc = false;

while (!eoc)

{

key\_ar = Key\_ar(j, arPhone);

if (String.Compare(key\_ar, key\_tmp) <= 0)

eoc = true;

else

{

arPhone[j + 1] = arPhone[j];

j = j - 1;

if (j < 0)

eoc = true;

}

}

arPhone[j + 1] = tmpPhone;

}

}

public void Sort7()

{

int i, j;

int flen = arPhone.Length;

bool eoc;

string key\_tmp, key\_ar;

FileMngmC.TbPhone tmpPhone;

for (i = (5\*flen / 8)+1; i < (7\*flen / 8); i++)

{

tmpPhone = arPhone[i];

key\_tmp = Key\_tmp(tmpPhone);

j = i - 1;

eoc = false;

while (!eoc)

{

key\_ar = Key\_ar(j, arPhone);

if (String.Compare(key\_ar, key\_tmp) <= 0)

eoc = true;

else

{

arPhone[j + 1] = arPhone[j];

j = j - 1;

if (j < 0)

eoc = true;

}

}

arPhone[j + 1] = tmpPhone;

}

}

**Протокол отладки:**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Исходные данные | Результат | Ошибка |
| 1 |  |  | Нет ошибок |
| 2 |  |  | Нет ошибок |
| 3 |  |  | Нет ошибок |
| 4 |  |  | Ошибка: введены неверные данные |
| 5 |  |  | Ошибка: введены неверные данные |
| 6 |  |  | Нет ошибок |