УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №4.2

по предмету

Основы алгоритмизации и программирования

Вариант 3

Выполнил:

Бетеня К.С.

Проверила:

Данилова Г.В.

Группа 351005

Минск 2024

Задание:

Задано число А и два вектора b[1..n] и c[1..n]. Найти множество I, являющееся подмножеством множества {1,...,n}, такое, что , а является максимальной из всех возможных. Написать рекурсивную процедуру.

Код программы на **Delphi**:

**MainUnit.pas**

Unit MainUnit;

Interface

Uses

Winapi.Windows,

Winapi.Messages,

System.SysUtils,

System.Variants,

System.Classes,

System.ImageList,

Vcl.Graphics,

Vcl.Controls,

Vcl.Forms,

Vcl.Dialogs,

Vcl.Menus,

Vcl.StdCtrls,

Vcl.Mask,

Vcl.ExtCtrls,

Vcl.Grids,

Vcl.ImgList,

Clipbrd;

Type

TLabeledEdit = Class(Vcl.ExtCtrls.TLabeledEdit)

Public

Procedure WMPaste(Var Msg: TMessage); Message WM\_PASTE;

End;

TMainForm = Class(TForm)

OpenDialog: TOpenDialog;

SaveDialog: TSaveDialog;

MainMenu: TMainMenu;

TaskLabel: TLabel;

FileButton: TMenuItem;

OpenButton: TMenuItem;

SaveButton: TMenuItem;

ExitButton: TMenuItem;

InstractionButton: TMenuItem;

AboutEditorButton: TMenuItem;

DemarcationLine: TMenuItem;

ALabeledEdit: TLabeledEdit;

NLabeledEdit: TLabeledEdit;

ALabel: TLabel;

NLabel: TLabel;

BVectorLabel: TLabel;

CVectorLabel: TLabel;

BVectorStringGrid: TStringGrid;

CVectorStringGrid: TStringGrid;

ResultButton: TButton;

ResultLabel: TLabel;

MainMenuImageList: TImageList;

Procedure AboutEditorButtonClick(Sender: TObject);

Procedure InstractionButtonClick(Sender: TObject);

Procedure FormCreate(Sender: TObject);

Procedure ALabeledEditContextPopup(Sender: TObject; MousePos: TPoint; Var

Handled: Boolean);

Procedure NLabeledEditContextPopup(Sender: TObject; MousePos: TPoint; Var

Handled: Boolean);

Procedure ALabeledEditKeyPress(Sender: TObject; Var Key: Char);

Procedure ALabeledEditKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Procedure NLabeledEditKeyPress(Sender: TObject; Var Key: Char);

Procedure NLabeledEditKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Procedure ALabeledEditChange(Sender: TObject);

Procedure NLabeledEditChange(Sender: TObject);

Procedure BVectorStringGridKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Procedure BVectorStringGridKeyPress(Sender: TObject; Var Key: Char);

Procedure CVectorStringGridKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Procedure CVectorStringGridKeyPress(Sender: TObject; Var Key: Char);

Procedure OpenButtonClick(Sender: TObject);

Procedure SaveButtonClick(Sender: TObject);

Procedure ResultButtonClick(Sender: TObject);

Procedure FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Procedure ExitButtonClick(Sender: TObject);

Function FormHelp(Command: Word; Data: NativeInt; Var CallHelp: Boolean): Boolean;

Private

Procedure WMGetMinMaxInfo(Var Msg: TWMGetMinMaxInfo);

Public

{ Public declarations }

End;

Const

MAIN\_CASE = ['0' .. '9', #08, #$16, '-'];

SERVICE\_CASE = [#08, #$16];

STRGRID\_CASE = ['0' .. '9', '-'];

DELETE\_KEY: Char = #127;

BACKSPACE\_KEY: Char = #08;

NULL\_POINT: Char = #0;

ZERO\_KEY: Char = '0';

MINUS\_KEY: Char = '-';

MIN\_INT\_NUM = -20\_000\_000;

MAX\_INT\_NUM = +20\_000\_000;

MIN\_N = 1;

MAX\_N = 13;

MAX\_COORD\_LENGTH: Integer = 9;

Var

MainForm: TMainForm;

IfDataSavedInFile: Boolean = False;

ResultString: String = #0;

Implementation

{$R \*.dfm}

Uses

AboutEditorUnit,

InstractionUnit,

BackendUnit,

FrontendUnit;

Procedure TMainForm.ALabeledEditChange(Sender: TObject);

Begin

LabelEditChange(ALabeledEdit.Text, NLabeledEdit.Text, BVectorStringGrid,

CVectorStringGrid);

End;

Procedure TMainForm.ALabeledEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Begin

Handled := True;

End;

Procedure TMainForm.ALabeledEditKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Var

CurKey: Char;

Begin

If (Key = VK\_DELETE) And (ALabeledEdit.SelText = '') Then

Begin

CurKey := IsCorrectDelete(DELETE\_KEY, ALabeledEdit.Text,

ALabeledEdit.SelStart + 1, MAX\_INT\_NUM,

MIN\_INT\_NUM);

If CurKey = NULL\_POINT Then

Key := 0;

End;

If (Key = VK\_DELETE) And (ALabeledEdit.SelText <> '') Then

Begin

CurKey := IsCorrectSelDelete(DELETE\_KEY, ALabeledEdit.Text,

ALabeledEdit.SelText, ALabeledEdit.SelStart,

MAX\_INT\_NUM, MIN\_INT\_NUM);

If CurKey = NULL\_POINT Then

Key := 0;

End;

If (Key = VK\_DOWN) Or (Key = VK\_RETURN) Then

SelectNext(ActiveControl, True, True);

If Key = VK\_UP Then

SelectNext(ActiveControl, False, True);

End;

Procedure TMainForm.ALabeledEditKeyPress(Sender: TObject; Var Key: Char);

Begin

If (NLabeledEdit.SelText <> '') And CharInSet(Key, SERVICE\_CASE) Then

Key := IsCorrectSelTextInputWithKey(Key, NLabeledEdit.Text,

NLabeledEdit.SelText,

NLabeledEdit.SelStart, MAX\_N, MIN\_N);

If Not CharInSet(Key, MAIN\_CASE) Then

Key := NULL\_POINT;

If (Length(ALabeledEdit.Text) > 0) And (ALabeledEdit.Text[1] = MINUS\_KEY) And

(Key = MINUS\_KEY) Then

Key := NULL\_POINT;

If (Length(ALabeledEdit.Text) > 0) And (ALabeledEdit.SelStart = 0) And

(Key = ZERO\_KEY) Then

Key := NULL\_POINT;

If (Length(ALabeledEdit.Text) > 1) And (ALabeledEdit.Text = MINUS\_KEY) And

(ALabeledEdit.SelStart = 0) Then

Key := NULL\_POINT;

If (Length(ALabeledEdit.Text) > 1) And (ALabeledEdit.Text[1] = MINUS\_KEY) And

(Key = ZERO\_KEY) And (ALabeledEdit.SelStart = 1) Then

Key := NULL\_POINT;

If (ALabeledEdit.Text = ZERO\_KEY) And (ALabeledEdit.SelStart = 1) Then

Key := NULL\_POINT;

If (Length(ALabeledEdit.Text) > 2) And Not CharInSet(Key, SERVICE\_CASE) And

(ALabeledEdit.SelText = '') Then

Key := IsCorrectAddInNum(Key, ALabeledEdit.Text, ALabeledEdit.SelStart,

MAX\_INT\_NUM, MIN\_INT\_NUM);

If (Key = BACKSPACE\_KEY) And (ALabeledEdit.SelText = '') Then

Key := IsCorrectDelete(Key, ALabeledEdit.Text, ALabeledEdit.SelStart,

MAX\_INT\_NUM, MIN\_INT\_NUM);

If (Key = BACKSPACE\_KEY) And (ALabeledEdit.SelText <> '') Then

Key := IsCorrectSelDelete(Key, ALabeledEdit.Text, ALabeledEdit.SelText,

ALabeledEdit.SelStart, MAX\_INT\_NUM,

MIN\_INT\_NUM);

End;

Procedure TMainForm.BVectorStringGridKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

StringGridVkBack(Key, BVectorStringGrid);

If ((Shift = [SsCtrl]) And (Key = Ord('V'))) Or ((Shift = [SsShift]) And

(Key = VK\_INSERT)) Then

StGridAddClipboard(Clipboard.AsText, BVectorStringGrid,

BVectorStringGrid.Col, BVectorStringGrid.Row);

If (Key = VK\_DOWN) Or (Key = VK\_RETURN) Then

SelectNext(ActiveControl, True, True);

If Key = VK\_UP Then

SelectNext(ActiveControl, False, True);

ResultsVisible(VectorStringGridChange(StrToInt(NLabeledEdit.Text)));

End;

Procedure TMainForm.BVectorStringGridKeyPress(Sender: TObject; Var Key: Char);

Var

Col, Row: Integer;

Begin

Col := BVectorStringGrid.Col;

Row := BVectorStringGrid.Row;

Key := StringGridKeyPress(BVectorStringGrid, Key, Col, Row);

If (Key <> NULL\_POINT) Then

Begin

BVectorStringGrid.Cells[Col, Row] := BVectorStringGrid.Cells[Col, Row] +

Key;

ResultsVisible(VectorStringGridChange(StrToInt(NLabeledEdit.Text)));

End;

End;

Procedure TMainForm.CVectorStringGridKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

Begin

StringGridVkBack(Key, CVectorStringGrid);

If ((Shift = [SsCtrl]) And (Key = Ord('V'))) Or ((Shift = [SsShift]) And

(Key = VK\_INSERT)) Then

StGridAddClipboard(Clipboard.AsText, CVectorStringGrid,

CVectorStringGrid.Col, CVectorStringGrid.Row);

If (Key = VK\_DOWN) Or (Key = VK\_RETURN) Then

SelectNext(ActiveControl, True, True);

If Key = VK\_UP Then

SelectNext(ActiveControl, False, True);

ResultsVisible(VectorStringGridChange(StrToInt(NLabeledEdit.Text)));

End;

Procedure TMainForm.CVectorStringGridKeyPress(Sender: TObject; Var Key: Char);

Var

Col, Row: Integer;

Begin

Col := CVectorStringGrid.Col;

Row := CVectorStringGrid.Row;

Key := StringGridKeyPress(CVectorStringGrid, Key, Col, Row);

If (Key <> NULL\_POINT) Then

Begin

CVectorStringGrid.Cells[Col, Row] := CVectorStringGrid.Cells[Col, Row] +

Key;

ResultsVisible(VectorStringGridChange(StrToInt(NLabeledEdit.Text)));

End;

End;

Procedure TMainForm.ExitButtonClick(Sender: TObject);

Begin

MainForm.Close;

End;

Procedure TMainForm.FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Var

ResultKey: Integer;

Begin

ResultKey := Application.Messagebox('Вы уверены, что хотите закрыть оконное

приложение?', 'Выход', MB\_YESNO +

MB\_ICONQUESTION + MB\_DEFBUTTON2);

If ResultKey = ID\_NO Then

CanClose := False;

If ResultLabel.Visible And (ResultKey = ID\_YES) And Not IfDataSavedInFile Then

Begin

ResultKey := Application.Messagebox('Вы не сохранили результат. Хотите

сделать это?', 'Сохранение', MB\_YESNO

+ MB\_ICONQUESTION + MB\_DEFBUTTON2);

If ResultKey = ID\_YES Then

SaveButtonClick(Sender);

End;

End;

Procedure TMainForm.FormCreate(Sender: TObject);

Begin

ALabeledEdit.EditLabel.Caption := '';

ALabeledEdit.Hint := '[' + IntToStr(MIN\_INT\_NUM) + '; ' +

IntToStr(MAX\_INT\_NUM) + ']';

NLabeledEdit.EditLabel.Caption := '';

NLabeledEdit.Hint := '[' + IntToStr(MIN\_N) + '; ' + IntToStr(MAX\_N) + ']';

BVectorStringGrid.Hint := '[' + IntToStr(MIN\_INT\_NUM) + '; ' +

IntToStr(MAX\_INT\_NUM) + ']';

CVectorStringGrid.Hint := '[' + IntToStr(MIN\_INT\_NUM) + '; ' +

IntToStr(MAX\_INT\_NUM) + ']';

ResultLabel.Caption := '';

End;

Function TMainForm.FormHelp(Command: Word; Data: NativeInt; Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

FormHelp := False;

End;

Procedure TMainForm.InstractionButtonClick(Sender: TObject);

Begin

Application.CreateForm(TInstraction, Instraction);

Instraction.ShowModal;

End;

Procedure TMainForm.NLabeledEditChange(Sender: TObject);

Begin

LabelEditChange(ALabeledEdit.Text, NLabeledEdit.Text, BVectorStringGrid,

CVectorStringGrid);

End;

Procedure TMainForm.NLabeledEditContextPopup(Sender: TObject; MousePos: TPoint; Var Handled: Boolean);

Begin

Handled := True;

End;

Procedure TMainForm.NLabeledEditKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Var

CurKey: Char;

Begin

If (Key = VK\_DELETE) And (NLabeledEdit.SelText = '') Then

Begin

CurKey := IsCorrectDelete(DELETE\_KEY, NLabeledEdit.Text,

NLabeledEdit.SelStart + 1, MAX\_INT\_NUM,

MIN\_INT\_NUM);

If CurKey = NULL\_POINT Then

Key := 0;

End;

If (Key = VK\_DELETE) And (NLabeledEdit.SelText <> '') Then

Begin

CurKey := IsCorrectSelDelete(DELETE\_KEY, NLabeledEdit.Text,

NLabeledEdit.SelText, NLabeledEdit.SelStart,

MAX\_INT\_NUM, MIN\_INT\_NUM);

If CurKey = NULL\_POINT Then

Key := 0;

End;

If (Key = VK\_DOWN) Or (Key = VK\_RETURN) Then

SelectNext(ActiveControl, True, True);

If Key = VK\_UP Then

SelectNext(ActiveControl, False, True);

End;

Procedure TMainForm.NLabeledEditKeyPress(Sender: TObject; Var Key: Char);

Const

GOOD\_SECOND\_VALUE = ['0' .. '3'];

Begin

If (NLabeledEdit.SelText <> '') And Not CharInSet(Key, SERVICE\_CASE) Then

Key := IsCorrectSelTextInputWithKey(Key, NLabeledEdit.Text,

NLabeledEdit.SelText,

NLabeledEdit.SelStart, MAX\_N, MIN\_N);

If Not CharInSet(Key, MAIN\_CASE) Then

Key := NULL\_POINT;

If (Length(NLabeledEdit.Text) <> 0) And (NLabeledEdit.SelStart = 0) And

(NLabeledEdit.SelText = '') And (Key = '1') And

Not CharInSet(NLabeledEdit.Text[1], GOOD\_SECOND\_VALUE) Then

Key := NULL\_POINT;

If (NLabeledEdit.Text = '1') And Not(CharInSet(Key, GOOD\_SECOND\_VALUE) Or

CharInSet(Key, SERVICE\_CASE)) And (NLabeledEdit.SelText = '') Then

Key := NULL\_POINT;

If (NLabeledEdit.Text <> '1') And (NLabeledEdit.SelStart = 1) And

(Length(NLabeledEdit.Text) <> 0) And (Key <> BACKSPACE\_KEY) And

(NLabeledEdit.SelText = '') Then

Key := NULL\_POINT;

If (NLabeledEdit.Text = '') And (Key = ZERO\_KEY) And

(NLabeledEdit.SelText = '') Then

Key := NUll\_POINT;

If (NLabeledEdit.Text <> '') And (NLabeledEdit.SelStart = 0) And

Not(CharInSet(Key, SERVICE\_CASE) Or (Key = '1')) And

(NLabeledEdit.SelText = '') Then

Key := NULL\_POINT;

If (Key = BACKSPACE\_KEY) And (NLabeledEdit.SelText = '') Then

Key := IsCorrectDelete(Key, NLabeledEdit.Text, NLabeledEdit.SelStart,

MAX\_N, MIN\_N);

If (Key = BACKSPACE\_KEY) And (NLabeledEdit.SelText <> '') Then

Key := IsCorrectSelDelete(Key, NLabeledEdit.Text, NLabeledEdit.SelText,

NLabeledEdit.SelStart, MAX\_N, MIN\_N);

End;

Procedure TMainForm.OpenButtonClick(Sender: TObject);

Var

IsCorrect: Boolean;

Begin

Repeat

If OpenDialog.Execute() Then

Begin

IsCorrect := IsReadable(OpenDialog.FileName);

ReadFromFile(IsCorrect, OpenDialog.FileName);

If Not IsCorrect Then

MessageBox(0, 'Невозможен ввод из файла!', 'Ошибка',

MB\_ICONERROR);

End

Else

IsCorrect := True;

Until IsCorrect;

End;

Procedure TMainForm.ResultButtonClick(Sender: TObject);

Var

ACount, NSize: Integer;

BVector, CVector: TMassive;

Sums: TMatrix;

ISubset: TByteSet;

Begin

SetLength(Sums, 0, 0);

ISubset := [];

ACount := StrToInt(ALabeledEdit.Text);

NSize := StrToInt(NLabeledEdit.Text);

InputVectorFromGrid(BVector, BVectorStringGrid, NSize);

InputVectorFromGrid(CVector, CVectorStringGrid, NSize);

TreatmentData(BVector, CVector, ACount, NSize, Sums, ISubset);

ResultLabel.Caption := CreateResultString(ACount, NSize, BVector, CVector,

ISubset);

ResultLabel.Visible := True;

SaveButton.Enabled := True;

End;

Procedure TMainForm.SaveButtonClick(Sender: TObject);

Var

IsCorrect: Boolean;

Begin

Repeat

If SaveDialog.Execute Then

Begin

IsCorrect := IsWriteable(SaveDialog.FileName);

InputInFile(IsCorrect, SaveDialog.FileName);

If Not IsCorrect Then

MessageBox(0, 'Невозможна запись в файл!', 'Ошибка',

MB\_ICONERROR);

End

Else

IsCorrect := True;

Until IsCorrect;

End;

Procedure TMainForm.AboutEditorButtonClick(Sender: TObject);

Begin

Application.CreateForm(TAboutEditor, AboutEditor);

AboutEditor.ShowModal;

End;

Procedure TMainForm.WMGetMinMaxInfo(Var Msg: TWMGetMinMaxInfo);

Begin

Msg.MinMaxInfo.PtMaxSize.X := Width;

Msg.MinMaxInfo.PtMaxSize.Y := Height;

Msg.MinMaxInfo.PtMaxTrackSize.X := Width;

Msg.MinMaxInfo.PtMaxTrackSize.Y := Height;

Left := (Screen.Width - Width) Div 2;

Top := (Screen.Height - Height) Div 2;

End;

Procedure TLabeledEdit.WMPaste(Var Msg: TMessage);

Begin

If Clipboard.HasFormat(CF\_TEXT) Then

Begin

Try

If (MainForm.ActiveControl = MainForm.ALabeledEdit) And Not

IsCorrectNumClipboard(Clipboard.AsText,

MainForm.ALabeledEdit.SelStart, MAX\_INT\_NUM, MIN\_INT\_NUM) Then

Raise Exception.Create('A Clipboard Error!');

If (MainForm.ActiveControl = MainForm.NLabeledEdit) And Not

IsCorrectNumClipboard(Clipboard.AsText,

MainForm.NLabeledEdit.SelStart, MAX\_N, MIN\_N) Then

Raise Exception.Create('N Clipboard Error!');

Except

Exit;

End;

End;

Inherited;

End;

End.

**FrontendUnit.pas**

Unit FrontendUnit;

Interface

Uses

Winapi.Windows,

System.SysUtils,

Vcl.Grids;

Function IsCorrectAddInNum(Key: Char; CurentStr: String; SelStart: Integer; Const

MAX, MIN: Integer): Char;

Function IsCorrectNumClipboard(ClipbrdText: String; Cursor: Integer; Const MAX,

MIN: Integer): Boolean;

Function IsCorrectSelTextInputWithKey(Key: Char; CurentText, SelText: String;

SelStart: Integer; Const MAX, MIN: Integer):

Char;

Function IsCorrectDelete(Key: Char; CurentText: String; SelStart: Integer; Const

MAX, MIN: Integer): Char;

Function IsCorrectSelDelete(Key: Char; CurentText, SelText: String; SelStart:

Integer; Const MAX, MIN: Integer): Char;

Function TryToAdd(Key: Char; Str: String; SelPos: Integer; Const MaxPoint,

MinPoint: Integer): Boolean;

Procedure VectorGridPrepearing(ArrayGrid: TStringGrid; NumOfCols: Integer);

Procedure ResettingArray(ArrayGrid: TStringGrid; NumOfCols: Integer);

Procedure VectorsVisible(Appearance: Boolean);

Procedure ResultsVisible(Appearance: Boolean);

Procedure LabelEditChange(ALabeledEdit, NLabeledEdit: String; BVectorStringGrid,

CVectorStringGrid: TStringGrid);

Procedure StringGridVkBack(Key: Word; Var VectorStringGrid: TStringGrid);

Function StringGridKeyPress(Var VectorStringGrid: TStringGrid; Key: Char; Col,

Row: Integer): Char;

Function VectorStringGridChange(NSize: Integer): Boolean;

Procedure StGridAddClipboard(ClipBoardText: String; Var StGrid: TStringGrid; Const

Col, Row: Integer);

Implementation

Uses

MainUnit;

Function IsCorrectAddInNum(Key: Char; CurentStr: String; SelStart: Integer; Const

MAX, MIN: Integer): Char;

Begin

Insert(Key, CurentStr, SelStart + 1);

Try

If (StrToInt(CurentStr) > MAX) Or (StrToInt(CurentStr) < MIN) Then

Key := NULL\_POINT;

If (Length(CurentStr) > 1) And (CurentStr[1] = ZERO\_KEY) Then

Key := NULL\_POINT;

If (Length(CurentStr) > 1) And (CurentStr[1] = MINUS\_KEY) And

(CurentStr[2] = ZERO\_KEY) Then

Key := NULL\_POINT;

Except

Key := NULL\_POINT;

End;

IsCorrectAddInNum := Key;

End;

Function IsCorrectNumClipboard(ClipbrdText: String; Cursor: Integer; Const MAX,

MIN: Integer): Boolean;

Var

IsCorrect: Boolean;

BufStr, WorkStr: String;

Begin

If (MainForm.ActiveControl = MainForm.ALabeledEdit) Then

Begin

BufStr := MainForm.ALabeledEdit.Text;

MainForm.ALabeledEdit.ClearSelection;

WorkStr := MainForm.ALabeledEdit.Text;

End

Else

Begin

BufStr := MainForm.NLabeledEdit.Text;

MainForm.NLabeledEdit.ClearSelection;

WorkStr := MainForm.NLabeledEdit.Text;

End;

Insert(ClipbrdText, WorkStr, Cursor + 1);

Try

IsCorrect := Not((StrToInt(WorkStr) > MAX) Or (StrToInt(WorkStr) < MIN));

IsCorrect := IsCorrect And (Length(WorkStr) > 1)

And (WorkStr[1] = ZERO\_KEY);

Except

IsCorrect := False;

End;

If Not IsCorrect And (MainForm.ActiveControl = MainForm.ALabeledEdit) Then

MainForm.ALabeledEdit.Text := BufStr;

If Not IsCorrect And (MainForm.ActiveControl = MainForm.NLabeledEdit) Then

MainForm.NLabeledEdit.Text := BufStr;

IsCorrectNumClipboard := IsCorrect;

End;

Function CheckKeyCondition(CurentText: String; Key: Char; Const MAX, MIN:

Integer): Char;

Begin

Try

If CurentText <> '' Then

If (StrToInt(CurentText) > MAX) Or (StrToInt(CurentText) < MIN) Then

Key := NULL\_POINT;

If (Length(CurentText) > 1) And (CurentText[1] = ZERO\_KEY) Then

Key := NULL\_POINT;

Except

Key := NULL\_POINT;

End;

CheckKeyCondition := Key;

End;

Function IsCorrectDelete(Key: Char; CurentText: String; SelStart: Integer; Const

MAX, MIN: Integer): Char;

Begin

Delete(CurentText, SelStart, 1);

IsCorrectDelete := CheckKeyCondition(CurentText, Key, MAX, MIN);

End;

Function IsCorrectSelDelete(Key: Char; CurentText, SelText: String; SelStart:

Integer; Const MAX, MIN: Integer): Char;

Begin

Delete(CurentText, SelStart + 1, Length(SelText));

IsCorrectSelDelete := CheckKeyCondition(CurentText, Key, MAX, MIN);

End;

Function IsCorrectSelTextInputWithKey(Key: Char; CurentText, SelText: String;

SelStart: Integer; Const MAX, MIN: Integer):

Char;

Begin

Delete(CurentText, SelStart + 1, Length(SelText));

Insert(Key, CurentText, SelStart + 1);

IsCorrectSelTextInputWithKey := CheckKeyCondition(CurentText, Key, MAX, MIN);

End;

Function TryToAdd(Key: Char; Str: String; SelPos: Integer; Const MaxPoint,

MinPoint: Integer): Boolean;

Begin

Insert(Key, Str, SelPos + 1);

Try

TryToAdd := Not((StrToInt(Str) > MaxPoint) Or (StrToInt(Str) < MinPoint))

Except

TryToAdd := False;

End;

End;

Procedure VectorGridPrepearing(ArrayGrid: TStringGrid; NumOfCols: Integer);

Var

I: Integer;

Begin

ArrayGrid.Cells[0, 0] := '№';

ArrayGrid.Cells[0, 1] := 'Число';

For I := 1 To NumOfCols Do

Begin

ArrayGrid.Cells[I, 0] := IntToStr(I);

ArrayGrid.Cells[I, 1] := '';

End;

End;

Procedure ResettingArray(ArrayGrid: TStringGrid; NumOfCols: Integer);

Begin

ArrayGrid.ColCount := NumOfCols + 1;

VectorGridPrepearing(ArrayGrid, NumOfCols)

End;

Procedure VectorsVisible(Appearance: Boolean);

Begin

MainForm.BVectorStringGrid.Visible := Appearance;

MainForm.BVectorLabel.Visible := Appearance;

MainForm.CVectorStringGrid.Visible := Appearance;

MainForm.CVectorLabel.Visible := Appearance;

MainForm.ResultButton.Visible := Appearance;

MainForm.ResultButton.Enabled := False;

MainForm.ResultLabel.Visible := False;

MainForm.SaveButton.Enabled := False;

IfDataSavedInFile := False;

If Appearance And (StrToInt(MainForm.NLabeledEdit.Text) > 4) Then

Begin

MainForm.BVectorStringGrid.Height := 88;

MainForm.CVectorStringGrid.Height := 88;

End;

If Appearance And Not(StrToInt(MainForm.NLabeledEdit.Text) > 4) Then

Begin

MainForm.BVectorStringGrid.Height := 68;

MainForm.CVectorStringGrid.Height := 68;

End;

End;

Procedure ResultsVisible(Appearance: Boolean);

Begin

MainForm.ResultButton.Enabled := Appearance;

If Not Appearance Then

Begin

MainForm.ResultLabel.Visible := Appearance;

MainForm.SaveButton.Enabled := Appearance;

IfDataSavedInFile := False;

End;

End;

Function CheckСhanges(IntLabEdit: String): Boolean;

Begin

Try

StrToInt(IntLabEdit);

CheckСhanges := True;

Except

CheckСhanges := False;

End;

End;

Procedure LabelEditChange(ALabeledEdit, NLabeledEdit: String; BVectorStringGrid,

CVectorStringGrid: TStringGrid);

Var

IsVisible: Boolean;

Begin

IsVisible := CheckСhanges(ALabeledEdit) And CheckСhanges(NLabeledEdit);

VectorsVisible(IsVisible);

If IsVisible Then

Begin

ResettingArray(BVectorStringGrid, StrToInt(NLabeledEdit));

ResettingArray(CVectorStringGrid, StrToInt(NLabeledEdit));

End;

End;

Procedure StringGridVkBack(Key: Word; Var VectorStringGrid: TStringGrid);

Var

Col, Row: Integer;

Begin

Col := VectorStringGrid.Col;

Row := VectorStringGrid.Row;

If (Key = VK\_BACK) Then

VectorStringGrid.Cells[Col, Row] := Copy(VectorStringGrid.Cells[Col, Row],

1, Length(VectorStringGrid.Cells[Col,

Row]) - 1);

End;

Function StringGridKeyPress(Var VectorStringGrid: TStringGrid; Key: Char; Col, Row: Integer): Char;

Const

MAX\_COORD\_LENGTH: Integer = 8;

Var

MinCount: Integer;

Begin

MinCount := 0;

If (VectorStringGrid.Cells[Col, Row] = ZERO\_KEY) Then

Key := NULL\_POINT;

If (VectorStringGrid.Cells[Col, Row] <> '') And

(VectorStringGrid.Cells[Col, Row] <> '-') And

(StrToInt(VectorStringGrid.Cells[Col, Row]) = 0) And (Key = ZERO\_KEY) Then

Key := NULL\_POINT;

If (Key = MINUS\_KEY) And (VectorStringGrid.Cells[Col, Row] <> '') Then

Key := NULL\_POINT;

If (Key = ZERO\_KEY) And (VectorStringGrid.Cells[Col, Row] = MINUS\_KEY) Then

Key := NULL\_POINT;

If Not CharInSet(Key, STRGRID\_CASE) Then

Key := NULL\_POINT;

If (VectorStringGrid.Cells[Col, Row] <> '') And

Not(TryToAdd(Key, VectorStringGrid.Cells[Col, Row],

Length(VectorStringGrid.Cells[Col, Row]), MAX\_INT\_NUM, MIN\_INT\_NUM)) Then

Key := NULL\_POINT;

If ((VectorStringGrid.Cells[Col, Row] <> '') And

(VectorStringGrid.Cells[Col, Row][1] = MINUS\_KEY)) Or (Key = MINUS\_KEY)

Then

MinCount := 1;

If Length(VectorStringGrid.Cells[Col, Row]) = MAX\_COORD\_LENGTH + MinCount Then

Key := NULL\_POINT;

StringGridKeyPress := Key;

End;

Function VectorStringGridChange(NSize: Integer): Boolean;

Var

I: Integer;

Begin

Try

For I := 1 To NSize Do

Begin

StrToInt(MainForm.BVectorStringGrid.Cells[I, 1]);

StrToInt(MainForm.CVectorStringGrid.Cells[I, 1]);

If (MainForm.BVectorStringGrid.Cells[I, 1] = '') Or

(MainForm.CVectorStringGrid.Cells[I, 1] = '') Then

Raise Exception.Create('Clear Cell.');

End;

VectorStringGridChange := True;

Except

VectorStringGridChange := False;

End;

End;

Procedure StGridAddClipboard(ClipBoardText: String; Var StGrid: TStringGrid; Const

Col, Row: Integer);

Var

IsCorrect: Boolean;

Begin

Try

IsCorrect := Not((StrToInt(ClipboardText) > MAX\_INT\_NUM) Or

(StrToInt(ClipboardText) < MIN\_INT\_NUM));

IsCorrect := IsCorrect And (ClipboardText[1] <> ' ');

IsCorrect := IsCorrect And (ClipboardText[Length(ClipBoardText)] <> ' ');

Except

IsCorrect := False;

End;

If IsCorrect Then

StGrid.Cells[Col, Row] := IntToStr(StrToInt(ClipboardText));

End;

End.

**BackendUnit.pas**

Unit BackendUnit;

Interface

Uses

Vcl.Grids,

System.SysUtils;

Type

TMassive = Array Of Integer;

TMatrix = Array Of Array Of Integer;

TByteSet = Set Of Byte;

Function TryReadNum(Var TestFile: TextFile; Var ReadStatus: Boolean; MAX\_NUM: Integer; EndOfNums: Boolean): Integer;

Function CheckNum(Num, Max, Min: Integer): Boolean;

Function TryRead(Var TestFile: TextFile): Boolean;

Function IsReadable(FilePath: String): Boolean;

Procedure ReadFromFile(Var IsCorrect: Boolean; FilePath: String);

Function IsWriteable(FilePath: String): Boolean;

Procedure InputInFile(Var IsCorrect: Boolean; FilePath: String);

Procedure InputVectorFromGrid(Var Vector: TMassive; StringGrid: TStringGrid; N:

Integer);

Procedure AddToArray(Var Results: TMatrix; CurrentPath: TMassive; NSize,

CurentSum: Integer);

Procedure CheckingCVectorCondition(CurrentPath, CVector: TMassive; ACount, NSize:

Integer; Var Results: TMatrix);

Function IndexOf(BVector: TMassive; Count: Integer): Integer;

Procedure SearchSuitableAmo(SubArray, BVector, CVector: TMassive; ACount, NSize:

Integer; Var Results: TMatrix; CurrentPath: Tmassive =

Nil);

Procedure Swap(Var Matrix: TMatrix; I, J: Integer);

Function Partition(Var Matrix: TMatrix; Left, Right: Integer): Integer;

Procedure QuickSort(Var Matrix: TMatrix; Left, Right: Integer);

Procedure AddResToSubset(Sums: TMatrix; Var ISubset: TByteSet);

Procedure TreatmentData(BVector, CVector: TMassive; ACount, NSize: Integer; Var

Sums: TMatrix; Var ISubset: TByteSet);

Function CreateStringWithVector(Vector: TMassive): String;

Function CreateStringWithSet(ISubset: TByteSet): String;

Function CreateResultString(ACount, NSize: Integer; BVector, CVector: TMassive;

ISubset: TByteSet): String;

Implementation

Uses

MainUnit,

FrontendUnit;

Function TryReadNum(Var TestFile: TextFile; Var ReadStatus: Boolean; MAX\_NUM:

Integer; EndOfNums: Boolean): Integer;

Const

SPACE\_LIMIT: Integer = 4;

Var

EndOfNum: Boolean;

Character, BufChar: Char;

SpaceCounter, Num, MinCount: Integer;

Begin

Num := 0;

EndOfNum := False;

SpaceCounter := 0;

Character := NULL\_POINT;

BufChar := Character;

MinCount := 1;

While ReadStatus And Not(EndOfNum) And Not(EOF(TestFile)) Do

Begin

BufChar := Character;

Read(TestFile, Character);

ReadStatus := ReadStatus And Not((Character <> ' ') And Not((Character >

Pred('0')) And (Character < Succ('9'))) And

(Character <> #13) And (Character <> #10) And (Character <>

'-'));

If (Character = ' ') Then

Inc(SpaceCounter)

Else

SpaceCounter := 0;

ReadStatus := Not(SpaceCounter = SPACE\_LIMIT);

If (Character > Pred('0')) And (Character < Succ('9')) Then

Num := Num \* 10 + Ord(Character) - 48;

If (Character = '-') Then

MinCount := -1;

ReadStatus := ReadStatus And Not((Character = '-') And (BufChar <> ' ')

And (BufChar <> #0));

ReadStatus := ReadStatus And Not((Character = '-') And (MinCount <> -1));

EndOfNum := ((Character = ' ') Or (Character = #13)) And ((BufChar >

Pred('0')) And (BufChar < Succ('9')));

ReadStatus := ReadStatus And Not((Num = 0) And (Character > Pred('0')) And

(Character < Succ('9')));

ReadStatus := ReadStatus And Not(Num > MAX\_NUM);

End;

ReadStatus := ReadStatus And Not(EOF(TestFile) And Not EndOfNums);

If ReadStatus Then

Num := MinCount \* Num;

TryReadNum := Num;

End;

Function CheckNum(Num, Max, Min: Integer): Boolean;

Begin

CheckNum := Not((Num > MAX) Or (Num < MIN));

End;

Function TryRead(Var TestFile: TextFile): Boolean;

Var

BufA, BufN, BufCoord, I: Integer;

ReadStatus: Boolean;

Begin

ReadStatus := True;

BufA := TryReadNum(TestFile, ReadStatus, MAX\_INT\_NUM, False);

ReadStatus := CheckNum(BufA, MAX\_INT\_NUM, MIN\_INT\_NUM);

BufN := TryReadNum(TestFile, ReadStatus, MAX\_N, False);

ReadStatus := CheckNum(BufN, MAX\_N, MIN\_N);

For I := 1 To BufN Do

Begin

BufCoord := TryReadNum(TestFile, ReadStatus, MAX\_INT\_NUM, False);

ReadStatus := CheckNum(BufCoord, MAX\_INT\_NUM, MIN\_INT\_NUM);

End;

For I := 1 To BufN Do

Begin

BufCoord := TryReadNum(TestFile, ReadStatus, MAX\_INT\_NUM, I = BufN);

ReadStatus := CheckNum(BufCoord, MAX\_INT\_NUM, MIN\_INT\_NUM);

End;

ReadStatus := ReadStatus And SeekEOF(TestFile);

TryRead := ReadStatus;

End;

Function IsReadable(FilePath: String): Boolean;

Var

TestFile: TextFile;

Begin

Try

AssignFile(TestFile, FilePath, CP\_UTF8);

Try

Reset(TestFile);

IsReadable := TryRead(TestFile);

Finally

Close(TestFile);

End;

Except

IsReadable := False;

End;

End;

Procedure ReadingProcess(Var IsCorrect: Boolean; Var MyFile: TextFile);

Var

I, A, N, Coord: Integer;

Begin

Try

Read(MyFile, A);

MainForm.ALabeledEdit.Text := IntToStr(A);

Read(MyFile, N);

MainForm.NLabeledEdit.Text := IntToStr(N);

VectorsVisible(True);

For I := 1 To N Do

Begin

Read(MyFile, Coord);

MainForm.BVectorStringGrid.Cells[I, 1] := IntToStr(Coord);

End;

For I := 1 To N Do

Begin

Read(MyFile, Coord);

MainForm.CVectorStringGrid.Cells[I, 1] := IntToStr(Coord);

End;

IsCorrect := True;

Except

IsCorrect := False;

End;

ResultsVisible(VectorStringGridChange(StrToInt(MainForm.NLabeledEdit.Text)));

IsCorrect := IsCorrect And SeekEOF(MyFile);

End;

Procedure ReadFromFile(Var IsCorrect: Boolean; FilePath: String);

Var

MyFile: TextFile;

Begin

If IsCorrect Then

Begin

AssignFile(MyFile, FilePath);

Try

Reset(MyFile);

Try

ReadingProcess(IsCorrect, MyFile);

Finally

Close(MyFile);

End;

Except

IsCorrect := False;

End;

End;

End;

Function IsWriteable(FilePath: String): Boolean;

Var

TestFile: TextFile;

Begin

Try

AssignFile(TestFile, FilePath);

Try

Rewrite(TestFile);

IsWriteable := True;

Finally

CloseFile(TestFile);

End;

Except

IsWriteable := False;

End;

End;

Procedure InputInFile(Var IsCorrect: Boolean; FilePath: String);

Var

MyFile: TextFile;

Begin

If IsCorrect Then

Begin

AssignFile(MyFile, FilePath, CP\_UTF8);

Try

ReWrite(MyFile);

Try

Writeln(MyFile, ResultString);

Finally

Close(MyFile);

End;

IfDataSavedInFile := True;

Except

IsCorrect := False;

End;

End;

End;

Procedure InputVectorFromGrid(Var Vector: TMassive; StringGrid: TStringGrid; N:

Integer);

Var

I: Integer;

Begin

SetLength(Vector, N);

For I := 1 To N Do

Vector[I - 1] := StrToInt(StringGrid.Cells[I, 1]);

End;

Procedure AddToArray(Var Results: TMatrix; CurrentPath: TMassive; NSize,

CurentSum: Integer);

Var

I: Integer;

Begin

SetLength(Results, Length(Results) + 1, NSize + 1);

For I := 0 To High(CurrentPath) Do

Results[High(Results)][I + 1] := CurrentPath[I] + 1;

Results[High(Results)][0] := CurentSum;

End;

Procedure CheckingCVectorCondition(CurrentPath, CVector: TMassive; ACount, NSize:

Integer; Var Results: TMatrix);

Var

CurentSum, I: Integer;

Begin

CurentSum := 0;

For I := 0 To High(CurrentPath) Do

CurentSum := CurentSum + CVector[CurrentPath[I]];

If (CurentSum <= ACount) Then

AddToArray(Results, CurrentPath, NSize, CurentSum);

End;

Function IndexOf(BVector: TMassive; Count: Integer): Integer;

Var

I, IndexNum: Integer;

Begin

IndexNum := -1;

For I := Low(BVector) To High(BVector) Do

If (BVector[I] = Count) And (IndexNum = -1) Then

IndexNum := I;

IndexOf := IndexNum;

End;

Procedure SearchSuitableAmo(SubArray, BVector, CVector: TMassive; ACount, NSize:

Integer; Var Results: TMatrix; CurrentPath: Tmassive =

Nil);

Var

CurrentPath2, CurrentPath1: TMassive;

I: Integer;

Begin

If (Length(SubArray) = 0) Then

CheckingCVectorCondition(CurrentPath, CVector, ACount, NSize, Results)

Else

Begin

SetLength(CurrentPath1, Length(CurrentPath) + 1);

For I := 0 To High(CurrentPath) Do

CurrentPath1[I] := CurrentPath[I];

CurrentPath1[Length(CurrentPath)] := IndexOf(BVector, SubArray[0]);

SearchSuitableAmo(Copy(SubArray, 1, Length(SubArray) - 1), BVector,

CVector, ACount, NSize, Results, CurrentPath1);

SetLength(CurrentPath2, Length(CurrentPath));

For I := 0 To High(CurrentPath) Do

CurrentPath2[I] := CurrentPath[I];

SearchSuitableAmo(Copy(SubArray, 1, Length(SubArray) - 1), BVector,

CVector, ACount, NSize, Results, CurrentPath2);

End;

End;

Procedure Swap(Var Matrix: TMatrix; I, J: Integer);

Var

Temp0, Temp1: Integer;

Begin

Temp0 := Matrix[I][0];

Temp1 := Matrix[I][1];

Matrix[I][0] := Matrix[J][0];

Matrix[I][1] := Matrix[J][1];

Matrix[J][0] := Temp0;

Matrix[J][1] := Temp1;

End;

Function Partition(Var Matrix: TMatrix; Left, Right: Integer): Integer;

Var

I, J, Pivot: Integer;

Begin

Pivot := Matrix[Left][0];

I := Left + 1;

For J := Left + 1 To Right Do

If (Matrix[J][0] > Pivot) Then

Begin

Swap(Matrix, I, J);

Inc(I);

End;

Swap(Matrix, Left, I - 1);

Partition := I - 1;

End;

Procedure QuickSort(Var Matrix: TMatrix; Left, Right: Integer);

Var

PivotIndex: Integer;

Begin

If Left < Right Then

Begin

PivotIndex := Partition(Matrix, Left, Right);

QuickSort(Matrix, Left, PivotIndex - 1);

QuickSort(Matrix, PivotIndex + 1, Right);

End;

End;

Procedure AddResToSubset(Sums: TMatrix; Var ISubset: TByteSet);

Var

I: Integer;

Begin

For I := 1 To High(Sums[0]) Do

If Sums[0][I] <> 0 Then

Include(ISubset, Sums[0][I]);

End;

Procedure TreatmentData(BVector, CVector: TMassive; ACount, NSize: Integer; Var

Sums: TMatrix; Var ISubset: TByteSet);

Begin

SearchSuitableAmo(BVector, BVector, CVector, ACount, NSize, Sums);

QuickSort(Sums, 0, Length(Sums) - 1);

If Sums <> Nil Then

AddResToSubset(Sums, ISubset);

End;

Function CreateStringWithVector(Vector: TMassive): String;

Var

VectorStr: String;

I: Integer;

Begin

VectorStr := '';

For I := 0 To High(Vector) Do

VectorStr := VectorStr + IntToStr(Vector[I]) + ' ';

CreateStringWithVector := VectorStr;

End;

Function CreateStringWithSet(ISubset: TByteSet): String;

Var

SetStr: String;

Curent: Byte;

Begin

SetStr := 'Подмножество I:';

If ISubset = [] Then

SetStr := SetStr + ' пустое множество...'

Else

Begin

For Curent In ISubset Do

SetStr := SetStr + ' ' + IntToStr(Curent);

End;

CreateStringWithSet := SetStr;

End;

Function CreateResultString(ACount, NSize: Integer; BVector, CVector: TMassive;

ISubset: TByteSet): String;

Var

ResultStr: String;

OutputStr: String;

Begin

ResultStr := 'A: ' + IntToStr(ACount) + ';'#13#10'N: ' + IntToStr(NSize) +

';'#13#10;

ResultStr := ResultStr + 'Вектор B: ' + CreateStringWithVector(BVector) +

#13#10;

ResultStr := ResultStr + 'Вектор C: ' + CreateStringWithVector(CVector) +

#13#10;

OutputStr := CreateStringWithSet(ISubset);

ResultStr := ResultStr + OutputStr;

ResultString := ResultStr;

CreateResultString := OutputStr;

End;

End.

Код программы на **C#**:

using System;

enum IOChoose : Byte

{

FILE = 1,

CONSOLE

}

class Proj4\_2

{

const int FILE\_VALUE = (int)IOChoose.FILE;

const int CONSOLE\_VALUE = (int)IOChoose.CONSOLE;

const int MIN\_INT\_NUM = -20\_000\_000;

const int MAX\_INT\_NUM = +20\_000\_000;

const int MIN\_N = 1;

const int MAX\_N = 15;

const int MIN\_FILE\_WAY\_SIZE = 4;

static void taskInfoOutput()

{

string outputString = $"""

The program determines a subset(I) of expenses {"{1..N}"}.

Where N is the number of coordinate vectors B and C.

The sum of the elements of vector B with numbers from

subset I is maximum, provided that the sum of the

elements from set C with the same numbers is not greater

than the integer A.

Restrictions:

1. A belongs to the interval [{MIN\_INT\_NUM}; {MAX\_INT\_NUM}];

2. N belongs to the interval [{MIN\_N}; {MAX\_N}];

3. Vector coordinates are in the interval [{MIN\_INT\_NUM};

{MAX\_INT\_NUM}];

""";

Console.WriteLine(outputString);

}

static void outputTextAboutIOSelection(string IOTextInfo)

{

string outputString = $"""

Select how you will {IOTextInfo} data:

{IOChoose.FILE}: {FILE\_VALUE} {IOChoose.CONSOLE}:

{CONSOLE\_VALUE}

Your option:

""";

Console.Write(outputString);

}

/// <summary>

/// Here you can write a file for what purposes you are using (input|output)

/// </summary>

/// <param name="IOTextInfo"></param>

/// <returns></returns>

static IOChoose chooseIOWay(string IOTextInfo)

{

outputTextAboutIOSelection(IOTextInfo);

IOChoose result = 0;

int ChosenPath = 0;

bool isCorrect = true;

do

{

isCorrect = true;

try

{

ChosenPath = Convert.ToInt32(Console.ReadLine());

}

catch

{

isCorrect = false;

}

switch (ChosenPath)

{

case FILE\_VALUE: result = IOChoose.FILE; break;

case CONSOLE\_VALUE: result = IOChoose.CONSOLE; break;

default: isCorrect = false; break;

}

if (!isCorrect) Console.Error.Write($"You should write one natural

number({FILE\_VALUE}|

{CONSOLE\_VALUE}): ");

else Console.WriteLine();

} while (!isCorrect);

return result;

}

static bool pathCondition(string filePath)

{

if (filePath.Length < MIN\_FILE\_WAY\_SIZE)

{

Console.Error.Write("The path is too short. Try again: ");

return false;

}

string bufstr = filePath.Substring(filePath.Length - MIN\_FILE\_WAY\_SIZE);

if (!bufstr.Equals(".txt"))

{

Console.Error.Write("Write .txt file. Try again: ");

return false;

}

return true;

}

static string inputFilePath()

{

string filePath = Console.ReadLine() ?? string.Empty;

while (!pathCondition(filePath))

filePath = Console.ReadLine() ?? string.Empty;

return filePath;

}

static bool isCanOpenFile(string filePath)

{

FileInfo fileInfo = new FileInfo(filePath);

return fileInfo.Exists;

}

static bool isCanWrite(string filePath)

{

try

{

using (StreamWriter writer = new StreamWriter(filePath))

writer.WriteLine(string.Empty);

return true;

}

catch

{

return false;

}

}

static bool isCanRead(string filePath)

{

try

{

using (StreamReader reader = new StreamReader(filePath))

reader.Read();

return true;

}

catch

{

return false;

}

}

static bool accessModifierControl(string accessModifier, string filePath)

{

bool resultModifier = true;

switch (accessModifier) {

case "input": resultModifier = isCanRead(filePath); break;

case "output": resultModifier = isCanWrite(filePath); break;

}

return resultModifier;

}

/// <summary>

/// Write "input" if you want to get the file path for input.

/// Write "output" if you want to get the path to the output file.

/// </summary>

/// <param name="accessModifier"></param>

/// <returns></returns>

static string inputPathToTheFile(string accessModifier)

{

string filePath = string.Empty;

bool isCorrect = true;

do

{

filePath = inputFilePath();

isCorrect = accessModifierControl(accessModifier, filePath) && isCanOpenFile(filePath);

if (!isCorrect)

Console.Error.Write("Can't open a file. Try write another way: ");

} while (!isCorrect);

return filePath;

}

static int inputNumberFromFile(StreamReader inputReader, ref bool

isCorrectInput, int MIN\_NUM, int MAX\_NUM)

{

int num = 0;

bool isCorrect = true;

int character, bufChar = 0;

while (isCorrect && isCorrectInput && (character = inputReader.Read()) != -1)

{

bool isServiceSymbol = character == ' ' || character == '\r' ||

character == '\n';

if (!(isServiceSymbol || !(character > '9' && character < '0')))

isCorrectInput = false;

if (!isServiceSymbol && isCorrectInput)

num = num \* 10 + character - 48;

if (bufChar != 0 && isCorrectInput && isServiceSymbol)

isCorrect = false;

if (num > MAX\_NUM)

isCorrect = false;

bufChar = character;

}

if (isCorrectInput && (num > MAX\_NUM || num < MIN\_NUM || bufChar == 0))

isCorrectInput = false;

return num;

}

static void inputVectorFromFile(StreamReader inputReader, ref int[] Vector,

ref bool isCorrectInput)

{

for (int i = 0; i < Vector.Length && isCorrectInput; ++i)

Vector[i] = inputNumberFromFile(inputReader, ref isCorrectInput,

MIN\_INT\_NUM, MAX\_INT\_NUM);

}

static bool isProcesOfFileInputCorrect(ref int ACount, ref int[] BVector,

ref int[] CVector, string filePath, ref

int NSize)

{

bool isCorrectInput = true;

using (StreamReader inputReader = new StreamReader(filePath))

{

ACount = inputNumberFromFile(inputReader, ref isCorrectInput,

MIN\_INT\_NUM, MAX\_INT\_NUM);

NSize = inputNumberFromFile(inputReader, ref isCorrectInput, MIN\_N,

MAX\_N);

BVector = new int[NSize];

CVector = new int[NSize];

inputVectorFromFile(inputReader, ref BVector, ref isCorrectInput);

inputVectorFromFile(inputReader, ref CVector, ref isCorrectInput);

isCorrectInput = isCorrectInput && inputReader.EndOfStream ? true :

false;

if (!isCorrectInput) Console.Error.WriteLine("Error in reading.

Try again.");

}

return isCorrectInput;

}

static void InputFormFile(ref int ACount, ref int[] BVector,

ref int[] CVector, ref int NSize)

{

string filePath = string.Empty;

do

{

Console.Write("Write way to your file (\*.txt): ");

filePath = inputPathToTheFile("input");

} while (!isProcesOfFileInputCorrect(ref ACount, ref BVector, ref CVector,

filePath, ref NSize));

}

static int InputNumberFromConsole(int MIN\_NUM, int MAX\_NUM)

{

int num = 0;

bool isCorrect = true;

do

{

isCorrect = true;

try

{

num = Convert.ToInt32(Console.ReadLine());

} catch

{

isCorrect = false;

}

isCorrect = isCorrect && !(num < MIN\_NUM) && !(num > MAX\_NUM) ? true :

false;

if (!isCorrect) Console.Error.Write($"Number must be from {MIN\_NUM} to

{MAX\_NUM}: ");

} while (!isCorrect);

return num;

}

static void InputVectorFromConsole(ref int[] Vector)

{

for (int i = 0; i < Vector.Length; ++i)

{

Console.Write($"Write your {i + 1} coordinate: ");

Vector[i] = InputNumberFromConsole(MIN\_INT\_NUM, MAX\_INT\_NUM);

}

}

static void InputFromConsole(ref int ACount, ref int[] BVector,

ref int[] CVector, ref int NSize)

{

Console.Write("Write A: ");

ACount = InputNumberFromConsole(MIN\_INT\_NUM, MAX\_INT\_NUM);

Console.Write("Write N - size of vectors: ");

NSize = InputNumberFromConsole(MIN\_N, MAX\_N);

BVector = new int[NSize];

Console.WriteLine("\nWrite vector B.");

InputVectorFromConsole(ref BVector);

CVector = new int[NSize];

Console.WriteLine("\nWrite vector C.");

InputVectorFromConsole(ref CVector);

Console.WriteLine();

}

static void inputData(ref int ACount, ref int[] BVector, ref int[] CVector,

ref int NSize)

{

IOChoose path = chooseIOWay("input");

switch (path)

{

case IOChoose.FILE: InputFormFile(ref ACount, ref BVector,

ref CVector, ref NSize); break;

case IOChoose.CONSOLE: InputFromConsole(ref ACount, ref BVector,

ref CVector, ref NSize);

break;

}

}

static int[][] AddToArray(int[][] array, int[] item, int NSize, int curentSum)

{

int[] newItem = new int[NSize + 1];

for (int i = 0; i < item.Length; ++i)

newItem[i + 1] = item[i] + 1;

newItem[0] = curentSum;

int[][] newArray = new int[array.Length + 1][];

Array.Copy(array, newArray, array.Length);

newArray[array.Length] = newItem;

return newArray;

}

static void checkingCVectorCondition(int[] currentPath, int[] CVector,

int ACount, ref int[][] results,

int NSize)

{

int curentSum = 0;

foreach (int i in currentPath)

curentSum += CVector[i];

if (curentSum <= ACount)

results = AddToArray(results, currentPath, NSize, curentSum);

}

static int[][] calculateSums(int[] BVector, int[] CVector, int ACount,

int NSize)

{

int[][] results = new int[0][];

void searchSuitableAmo(int[] subArray, int[] currentPath)

{

if (subArray.Length == 0)

checkingCVectorCondition(currentPath, CVector, ACount,

ref results, NSize);

else

{

int[] newPath1 = new int[currentPath.Length + 1];

Array.Copy(currentPath, newPath1, currentPath.Length);

newPath1[currentPath.Length] = Array.IndexOf(BVector,

subArray[0]);

searchSuitableAmo(subArray[1..], newPath1);

int[] newPath2 = new int[currentPath.Length];

Array.Copy(currentPath, newPath2, currentPath.Length);

searchSuitableAmo(subArray[1..], newPath2);

}

}

searchSuitableAmo(BVector, new int[0]);

return results;

}

static void swap(int[][] array, int i, int j)

{

int temp0 = array[i][0];

int temp1 = array[i][1];

array[i][0] = array[j][0];

array[i][1] = array[j][1];

array[j][0] = temp0;

array[j][1] = temp1;

}

static int partition(int[][] array, int left, int right)

{

int pivot = array[left][0];

int i = left + 1;

for (int j = left + 1; j <= right; j++)

{

if (array[j][0] > pivot)

{

swap(array, i, j);

i++;

}

}

swap(array, left, i - 1);

return i - 1;

}

static void quickSort(int[][] array, int left, int right)

{

if (left < right)

{

int pivotIndex = partition(array, left, right);

quickSort(array, left, pivotIndex - 1);

quickSort(array, pivotIndex + 1, right);

}

}

static void addResToSubset(int[][] sums, ref HashSet<int> ISubset)

{

for (int i = 1; sums.Length != 0 && i < sums[0].Length; i++)

if (sums[0][i] != 0)

ISubset.Add(sums[0][i]);

}

static void treatmentData(int ACount, int[] BVector, int[] CVector, int NSize,

ref HashSet<int> ISubset)

{

int[][] sums = calculateSums(BVector, CVector, ACount, NSize);

quickSort(sums, 0, sums.Length - 1);

addResToSubset(sums, ref ISubset);

}

static string createStringWithVector(int[] Vector)

{

string vectorStr = string.Empty;

for (int i = 0; i < Vector.Length; i++)

vectorStr += Vector[i] + " ";

return vectorStr;

}

static string createStringWithSet(HashSet<int> ISubset)

{

if (ISubset.Count == 0)

return " empty set...";

string setStr = string.Empty;

foreach (int i in ISubset)

setStr += " " + i;

return setStr;

}

static string createResultString(int ACount, int[] BVector, int[] CVector,

int NSize, HashSet<int> ISubset)

{

string resultStr = $"""

A: {ACount};

N: {NSize};

Vector B: {createStringWithVector(BVector)}

Vector C: {createStringWithVector(CVector)}

Result set I:{createStringWithSet(ISubset)}

""";

return resultStr;

}

static bool isProcesOfFileOutputCorrect(string filePath, string resultStr)

{

try

{

using (StreamWriter writerOutput = new StreamWriter(filePath))

writerOutput.WriteLine(resultStr);

Console.WriteLine("Data successfully written to file.");

return true;

}

catch

{

Console.Error.WriteLine("Error in writing. Try again.");

return false;

}

}

static void OutputFormFile(string resultStr)

{

string filePath = string.Empty;

do

{

Console.Write("Write way to your file (\*.txt): ");

filePath = inputPathToTheFile("output");

} while (!isProcesOfFileOutputCorrect(filePath, resultStr));

}

static void OutputFromConsole(string resultStr)

{

Console.WriteLine($"{resultStr}");

}

static void outputData(int ACount, int[] BVector, int[] CVector, int NSize,

HashSet<int> ISubset)

{

string resultStr = createResultString(ACount, BVector, CVector, NSize,

ISubset);

IOChoose path = chooseIOWay("output");

switch (path)

{

case IOChoose.FILE: OutputFormFile(resultStr); break;

case IOChoose.CONSOLE: OutputFromConsole(resultStr); break;

}

}

public static void Main(string[] args)

{

int ACount = 0;

int NSize = 0;

int[] BVector = { };

int[] CVector = { };

HashSet<int> ISubset = new HashSet<int>();

taskInfoOutput();

inputData(ref ACount, ref BVector, ref CVector, ref NSize);

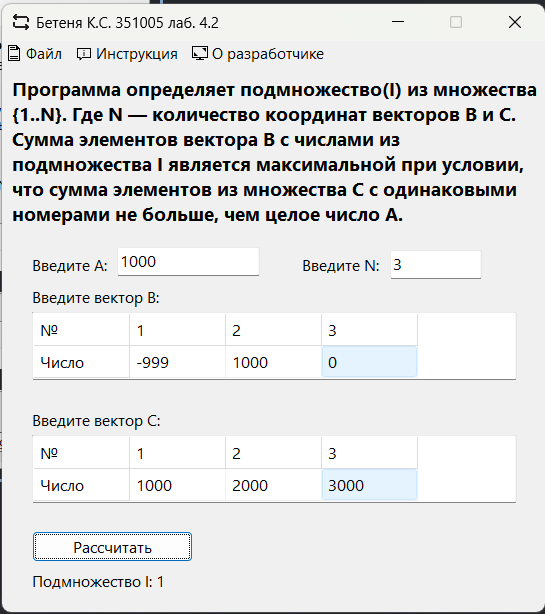
treatmentData(ACount, BVector, CVector, NSize, ref ISubset);

outputData(ACount, BVector, CVector, NSize, ISubset);

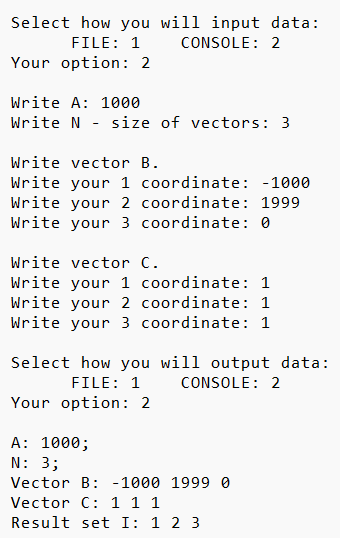
}

}

Результат в **Delphi**:



Результат в **C#**:



Блок-схема:

