{
Task Analysis for LGame
---- -------- --- -----

The obvious problem in this exercise is the large "dictionary" file. It's too big to read into memory (unless you know how to use the machine's extra memory). Since you have to output all the words and combinations with highest score, the entire file has to be read at least once. This extra requirement was in fact a small hint to help you find the highest scoring word when no exact match can be found. As reading a large file takes quite some time the obvious strategy is to avoid reading the file more than once. The purpose of the exercise was not, as some solutions seemed to suggest, to find a way to cram all the words in memory, nor was it an exercise in making indexes or using hashing procedures. The fact that the word list could vary from run to run was meant as a hint that elaborate pre-processing of the list should be avoided as it would be used only once. I shudder to think what could have happened if the list we offered would not have been sorted. Not that it mattered for this problem, but probably people would have felt the need to sort it (exceeding the time limit). This time limit was also a hint to put you on the right track, i.e., it would be a waste of valuable time to process the word list more than once. However, some of you were so obsessed by it that you deemed it necessary to put a timer in the program, which, in some cases, let to disaster.

After seeing some of the solutions you came up with I was quite amazed to see how some of you can turn a perfectly simple problem (remember, it was the first program on the second day, which tells you something about its complexity (or more accurately lack of complexity) into extremely complex programming. Still there remain several ways to find the highest scoring word(s) or pair(s) of words given a set of 3 to 7 letters, but there are certain inherent conditions you can use to avoid extra work.

- With an exact match (i.e. the letters in the wordlist-word match the letters given) you will always score maximum points. Once you have found 1 exact match you do not have to look at words with lesser characters unless the number of letters given is 7 (then you have to keep track of 4 and 3 letter words [sic]) or 6 (then you have to keep track of 3 letter words).

- It's a waste of time and space to store all 3 and 4 letter words. Only when necessary (see 1) you should store those 3 and 4 letter words that match with the letters given. These words can be stored in arrays. The number of unique combinations of 3 and 4 letters given a set of 7 letters is 35 (for 3) and 35 (for 4). The words are not unique so reserving 3*2*35 and 4*3*2*35 will cover all possibilities amply (in fact this is far too much as they are actual words, and most letterpermutitions do not yield grammatical words). After reading all the words in the wordlist you can easily check for combinations. You do not have to count their scores separately as they are exact matches and therefore score maximum points.

- The fun starts when no exact match can be found. When 6 or 7 letters were given finding a combination of 3+3 or 4+3 will again give maximum score, but you can't know that a combination will be found so you have

to keep track of words of 5 letters as well.

- When no exact match can be found and no combinations can be found a word with fewer letters can score more points than a word with more letters.

- When no exact match can be found words with an equal number of letters should be checked for highest score.

Possible solution:

The way you check is crucial. An alphabetic check will mean extra time as each word has to be sorted and it only works with exact matches. Generating all possible combinations from the set of letters and matching against these takes less time and will ensure that all possible words are found but it means that lacking an exact match each word read from the word list has to be compared with at least 35 and at most 95 combinations. A one-on-one check should therefore be preferred. Eliminating the letters of the word read against those in the set of letters seems to work best. Storing the highest scoring word(s) so far, while processing the wordlist, will solves 3, 4 and 5 from the list above. That way you will always have a solution (at least when a word can be found with the letters given).

How to speed up?

- read the word list only once

- use string operations when comparing

- use recursion when eliminating

- use dynamic data structures to store possible candidates and to store 3 and 4 letter words. That way you do not have to initialise arrays and you can sort on value while storing and you do not have to figure out, before hand, how much memory-space you have to reserve for the arrays.

- do not check words unnecessarily (see above).

Connie Veugen
Scientific Committee IOI'95
}

```
program LGame;
{----- Solution using arrays, ample storage of variables    -----}
{----- File is only read once and only valid candidates     -----}
{-----                  are stored                          -----}
{-----               Connie Veugen 7 6 95                   -----}

const
        MaxWrdL   =         7;
        MaxThree  =    35*3*2;  {35 unique possibilties *  no of permutations}
        MaxFour   =  35*4*3*2;  {ibid, both are far more than a normal}
                                {letter distribution would yield}
        MaxCand   =     7*6*5;  {possible candidates}
                                {this margin is absurd as only the highest}
```

```
                              {values will be stored}

type
        WrdType = string[MaxWrdL+1];
        Candid  = record
                     Wrd : WrdType;
                            Pnt : word
                  end;
        Ar3     = array[1..MaxThree] of WrdType;
        Ar4     = array[1..MaxFour] of WrdType;
        ArCan   = array[1..MaxCand] of Candid;

var
        InFile,
        OutFile,
        WBook       : text;
        Letters     : WrdType;
        LengthThree : Ar3;      {storage for 3 letter words}
        LengthFour  : Ar4;      {storage for 4 letter words}
        MaxValue,               {highest score so far}
        CountThree,             {number of 3 letter words}
        CountFour,              {number of 4 letter words}
        CountCand  :  word;     {number of possible candidates}
        Found      : ArCan;     {word(s) (pair(s)) with highest score so far}


        procedure Init;

        var I : byte;

        begin {Init}
                CountThree := 0;
                CountFour := 0;
                CountCand  := 0;
                MaxValue  := 0;
                for I := 1 to MaxCand do
                begin
                        Found[I].Wrd := '';
                        Found[I].Pnt := 0;
                    end;
            assign(InFile, 'input.txt');
            assign(WBook, 'words.txt');
            assign(OutFile, 'output.txt');
            reset(InFile);
            reset(WBook);
            rewrite(OutFile)
        end; {Init}


        function Value(Wrd : WrdType) : byte;

    var
           Total, I : byte;

    begin {Value}
            Total := 0;
            for I := 1 to length(Wrd) do
```

```
            case upcase(Wrd[I]) of
                    'E','I','S'             : Total := Total + 1;
                            'A','N','R','T'         : Total := Total + 2;
                            'L','O'                 : Total := Total + 3;
                            'C','D','U'             : Total := Total + 4;
                            'B','G','H','M','P','Y': Total := Total + 5;
                            'F','K','V','W'         : Total := Total + 6;
                            'J','Q','X','Z'         : Total := Total + 7
            end {case};
          Value := Total
end; {Value}


function OK(Wrd1, Wrd2 : WrdType):boolean;
{-------------------------------------------------------}
{checks if there are no 'illegal' letters in the inputline}
{as only the letters from the set are alowed and they may }
{only be used once.                                       }
{-------------------------------------------------------}

var
       Good : boolean;
       I    : byte;

begin {OK}
    {no more letters in inputline, so all matched}
            if length(Wrd1) = 0 then OK := true else
            {no more letters in set, so no match}
            if length(Wrd2) = 0 then OK := false else
            {illegal letter in inputline}
                    if pos(Wrd1[1], Wrd2) = 0 then OK := false else
                    begin
                            {take the letter out of the set as it can
                                    only be used once}
                                    delete(Wrd2, pos(Wrd1[1],Wrd2), 1);
                            {take the letter out of the inputline as
                                    it has already been processed}
                                    delete(Wrd1, 1, 1);
                            {check what is left}
                                    OK := OK(Wrd1, Wrd2)
                            end
end; {OK}


procedure ProcessWBook(Let : WrdType; var Max, Count, A3, A4 : word);
{-------------------------------------------------------------------}
{read wordlist line by line and process each line as it is read     }
{do not proces lines that contain 'illegal' letters or that use a   }
{letter from the set more than once, only proces candidates with a  }
{higher or equal score than the maximum at the given moment, if the }
{set was 7 letters then store possible 3 and 4 letter candidates    }
{if it was 6 then store 3 letter candidates only.                   }
{-------------------------------------------------------------------}

var
       InputLine  : WrdType;
       LengthInpL,
```

```
           LengthLet,
           ValueCand : byte;
           Higher    : boolean;

    begin {ProcessWBook}
                while not eof(WBook) do
                begin
                        readln(WBook, InputLine);
                                Higher := false;
                {only check valid candidates}
                                if OK(InputLine, Let) then
                                begin
                                        ValueCand := Value(InputLine);
                                        LengthInpL := length(InputLine);
                                        LengthLet := length(Let);
                                        Higher := ValueCand >= Max;
                                        if Higher then Max := ValueCand;
                                        {only store those with maximum or
higher score}
                                                if (LengthInpL = LengthLet) or
Higher then
                                                        begin
                                                                inc(Count);
                                                                Found[Count].Wrd
:= InputLine;
                                                                Found[Count].Pnt
:= ValueCand
                                                        end;
                                        if LengthLet = 7 then
                                {check for possible 4}
                                                if LengthInpL = 4 then
                                                        begin
                                                        inc(A4);
                                                                LengthFour[A4]
:= InputLine
                                                        end;
                                        if LengthLet >= 6 then
                                {check for possible 3}
                                                if LengthInpL = 3 then
                                                        begin
                                                        inc(A3);
                                                                LengthThree[A3]
:= InputLine
                                                        end;
                                end
                        end;
                        close(WBook)
    end; {ProcessWBook}



    procedure StorePairs(Wrd1, Wrd2: WrdType; Score : word);
    {-------------------------------------------------------}
    {valid word pairs have to be stored in a particular way }
    {for output.                                            }
    {-------------------------------------------------------}
```

```
    var
           I : byte;
           Bingo : boolean;

    begin {StorePairs}
                I := 1;
                Bingo := false;
        {do not store the same pair twice}
                while not Bingo and (I <= MaxCand) and (Found[I].Wrd <> '') do
                       begin
                       if (Wrd1 + ' ' + Wrd2 = Found[I].Wrd) or
                               (Wrd2 + ' ' + Wrd1 = Found[I].Wrd) then
                                       Bingo := true
                               else inc(I)
                       end;
                if not Bingo then with Found[I] do
                begin
                       Wrd := Wrd1 + ' ' + Wrd2;
                       Pnt := Score
                       end
   end; {StorePairs}


    procedure In3and4(var Count, MaxV : word; A3, A4 : word; Let : WrdType);
    {-------------------------------------------------------------------------}
    { when the letterset consists of 7 letters you have to check for pairs }
    { of 3 and 4 letters.                                                     }
    {-------------------------------------------------------------------------}

    var
           I, J, Score : word;
           Temp        : WrdType;

    begin {In3and4}
           for I := 1 to A3 do
               for J := 1 to A4 do
                      begin
                   {make a dummy word out of the two candidates}
                          Temp := LengthThree[I] + LengthFour[J];
                             Score := Value(Temp);
                   {check if no double letters are used and if
                            this is a valid candidate scorewise}
                             if (OK(Temp, Let)) and (Score >= MaxV) then
                               begin
                                   inc(Count);
                                       if Score > MaxV then MaxV := Score;
                                       StorePairs(LengthThree[I],
LengthFour[J], Score)
                               end;
                     end
   end; {In3and4}


   procedure In3and3(var Count, MaxV : word; A3: word; Let : WrdType);
   {-------------------------------------------------------------}
   { when the set has 7 or 6 letters check for pais of 3 and 3      }
   { further comments see In4and3                                   }
```

```
   {-------------------------------------------------------------------}

   var
           I, J, Score : word;
       Temp        : WrdType;

    begin {In3and3}
           for I := 1 to A3 do
               for J := 1 to A3 do
                      begin
                           Temp := LengthThree[I] + LengthThree[J];
                               Score := Value(Temp);
                               if (OK(Temp, Let)) and (Score >= MaxV) then
                                 begin
                                   inc(Count);
                                       if Score > MaxV then MaxV := Score;
                               StorePairs(LengthThree[I], LengthThree[J], Score)
                                 end;
                     end
   end; {In3and3}


   procedure WriteOutput(Count, MaxV : word);

   var
           I    : byte;
           Temp : WrdType;

   begin {WriteOutput}
          writeln(OutFile, MaxV);
          for I := 1 to Count do
           {as there are older "high-scoring" candidates
             write only those with actual MaxV}
               if Found[I].Pnt = MaxV
                  then writeln(OutFile, Found[I].Wrd);
          close(OutFile)
   end;{WriteOutput}


begin {main}
        Init;
        readln(InFile, Letters);
        close(InFile);
        ProcessWBook(Letters, MaxValue, CountCand, CountThree, CountFour);
        if (length(Letters) = 7) and (CountThree <> 0) and (CountFour <> 0)
           then In3and4(CountCand, MaxValue, CountThree, CountFour, Letters);
        if (length(Letters) >= 6) and (CountThree <> 0)
           then In3and3(CountCand, MaxValue, CountThree, Letters);
        WriteOutput(CountCand, MaxValue)
end.
```