Games

Word Search Puzzles

	0	1	2	3
0	t	h	i	s
1	w	a	t	s
2	o	a	h	g
3	f	g	d	t

```
for each word W in the word list
  for each row R
    for each column C
      for each direction D
        check if W exists
            at row R, column C in direction D
```

#checks = W•R•C•8

THRijswijk

Games

Word Search Puzzles

```
for each row R
  for each column C
    for each direction D
      for each word length L
        check if L chars starting at row R
        column C in direction D form a word
```

#checks = R•C•8•L•^{2}log(W)

Stel:
R=32, C=32, W=234946 (Webster),
L=24 (formaldehydesulphoxylate)

Brute Force: #checks =
234946•32•32•8 = 1924677632

Binary Search: #checks =
32•32•8•24•^{2}log(234946) = 3538944
ruim **500x** sneller!

THRijswijk

STL

Binary Search

```
bool binary_search(
  ForwardIterator first, ForwardIterator last,
  const T& value);

ForwardIterator lower_bound(
  ForwardIterator first, ForwardIterator last,
  const T& value);
// geeft eerste plaats waar value ingevoegd
// zou kunnen worden.
ForwardIterator upper_bound(
  ForwardIterator first, ForwardIterator last,
  const T& value);
// geeft laatste plaats waar value ingevoegd
// zou kunnen worden.

pair<ForwardIterator, ForwardIterator>
equal_range(
  ForwardIterator first, ForwardIterator last,
  const T& value);
// geeft lower_bound en upper_bound
```

sequence moet gesorteerd zijn met less<T>
als anders gesorteerd: Comperator meegeven!

THRijswijk

Games

Word Search Puzzles

```
for each row R
  for each column C
    for each direction D
      for each word length L
        check if L chars starting at row R
        column C in direction D form a word
        if they do not form a prefix,
          break; // the innermost loop
```

Implementatie om te kijken of een woord bestaat en of het een prefix is.

```
vector<string>::const_iterator itr (
  lower_bound(
    theWords.begin(),
    theWords.end(), word));
if (itr==theWords.end() || !isPrefix(word, *itr))
    break;
```

THRijswijk

Word Search

Implementatie

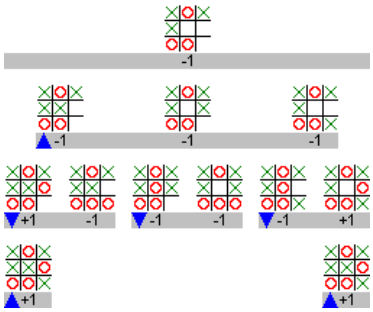
- gebruikt de matrix template (niet std) zie p. 111.
- isPrefix p. 396 kan eenvoudiger!
- toVec p. 394 kan eenvoudiger.
- Zie: http://bd.thrijswijk.nl/syso1

- SortStrings.cpp
  - sorteert woordenboek (was niet gesorteerd A en a stonden bij elkaar).
  - zoek maximale woordlengte met STL algoritme max_element
- GenPuzzle.cpp
  - genereert een random puzzle
  - kans is evenredig met aantal maal dat letter voorkomt in webster.txt
- **WordSrch.cpp**

THRijswijk

Games

Tic-Tac-Toe



backtracking algoritme met minimax strategie!

THRijswijk

minimax

```
int TicTacToe::chooseMove(Side s,
      int& bestRow, int& bestColumn) {
   Side opp(s==COMPUTER ?
                        HUMAN : COMPUTER);
   int value(s==COMPUTER ?
            HUMAN_WIN : COMPUTER_WIN);
   int simpleEval(positionValue());
   if (simpleEval!=UNCLEAR)
      return simpleEval;
   for (int r(0); r<board.numrows(); ++r)
      for (int c(0); c<board.numcols(); ++c)
         if (squareIsEmpty(r, c)) {
            place(r, c, s);
            int dc;
            int reply(chooseMove(opp, dc, dc));
            place(r, c, EMPTY);
            if (s==COMPUTER && reply>value ||
                s==HUMAN && reply<value) {
               value=reply;
               bestRow=r;
               bestColumn=c;
            }
         }
   return value;
}
```

THRijswijk

TTT

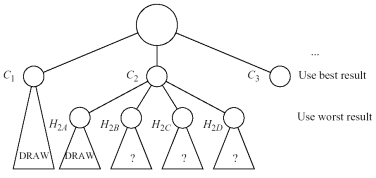
aantal aanroepen chooseMove bij eerste zet (computer begint)

- Maximaal: 1+9+9x8+9x8x7+ ... + 9x8x7x6x5x4x3x2x1 = 986410
- Stoppen als er een winnaar is = 549946
- Toepassen alpha-beta pruning = 18297
- Toepassen transposition table = 7954
- Zoek identieke stellingen (draaien en spiegelen) = ???

THRijswijk

Games

Alpha-Beta pruning



Zie p. 398.
After H_{2A} is evaluated, C_2, which is the minimum of the H_2's, is at best a draw. Consequently, it cannot be an improvement over C_1. We therefore do not need to evaluate H_{2B}, H_{2C}, and H_{2D}, and can proceed directly to C_3.

THRijswijk

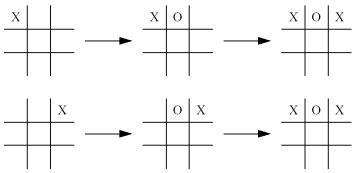
alpha-beta

```
int TicTacToe::chooseMove(Side s,
      int& bestRow, int& bestColumn,
      int alpha, int beta) {
   Side opp(s==COMPUTER ?
                  HUMAN : COMPUTER);
   int value(s==COMPUTER ? alpha : beta);
   int simpleEval(positionValue());
   if (simpleEval!=UNCLEAR) return simpleEval;
   for (int r(0); r<board.numrows(); ++r)
      for (int c(0); c<board.numcols(); ++c)
         if (squareIsEmpty(r, c)) {
            place(r, c, s); int dc;
            int reply(chooseMove(opp, dc, dc,
                                 alpha, beta));
            place(r, c, EMPTY);
            if (s==COMPUTER && reply>value ||
                s==HUMAN && reply<value) {
               value=reply;
               if (s==COMPUTER) alpha=value;
               else beta=value;
               bestRow=r; bestColumn=c;
               if (alpha>=beta) return value;
            }
         }
   return value;
}
```

THRijswijk

Games

Transpostions



Zie p. 400.
Two searches that arrive at identical positions.

THRijswijk

transpositions

```
class Position {
public:
   Position(const matrix<int>& theBoard):
         board(theBoard) {
   }
   bool operator<(const Position& rhs) const;
private:
   matrix<int> board;
};

bool Position::operator<(const Position & rhs)
      const {
   for (int i(0); i<board.numrows(); ++i)
      for (int j(0); j<board.numcols(); ++j)
         if (board[i][j]!=rhs.board[i][j])
            return board[i][j]<rhs.board[i][j];
   return false;
}

class TicTacToe {
//...
private:
   map<Position, int> transpositions;
};
```

THRijswijk

transpositions

```
int TicTacToe::chooseMove(Side s,
      int& bestRow, int& bestColumn,
      int alpha, int beta, int depth) {
   Position thisPosition(board);
   if (depth>=3 && depth<=5 ) {
      MapItr itr(transpositions.find(thisPosition));
      if (itr!=transpositions.end())
         return (*itr).second;
   }
// idem ...
            int reply(chooseMove(opp, dc, dc,
                                 alpha, beta, depth+1));
// idem ...
            if (alpha>=beta) goto Done;
         }
      }
Done:
   if (depth<=MAX_TABLE_DEPTH )
      transpositions[thisPosition]=value;
   return value;
}
```

THRijswijk

Utilities

File Compression

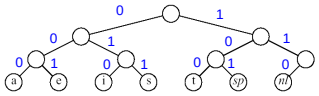
- 100 % reproduceerbaar:
  - Huffman codering (variabele length coding) maakt gebruik van frequentieverdeling codes
  - Ziv-Lempel codering (fixed length coding) zoekt meest voorkomende substrings.
- niet 100% reproduceerbaar:
  - jpg
    - maak pixels die bijna gelijk zijn gelijk
  - mp3
    - verwijder overstemde geluiden
  - divx
    - sla alleen verschillen met vorige frame op

THRijswijk

Codering

Fixed length

Character	Code	Frequency	Total Bits
a	000	10	30
e	001	15	45
i	010	12	36
s	011	3	9
t	100	4	12
sp	101	13	39
nl	110	1	3
Total			174

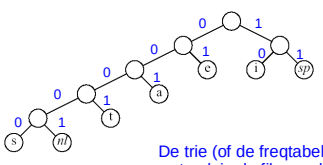


THRijswijk

Codering

Variable length

Character	Code	Frequency	Total Bits
a	001	10	30
e	01	15	30
i	10	12	24
s	00000	3	15
t	0001	4	16
sp	11	13	26
nl	00001	1	5
Total			146



De trie (of de freqtabel) moet ook in de file worden opgenomen!

THRijswijk

Huffman algoritme

Bepaalt een optimale coderings trie voor een bepaalde tekst.

- Bepaal de frequentie van elk karakter in de tekst.
- Maak van elk karakter een boom (= blad) met als gewicht de frequentie van dit karakter in de tekst.
- Voeg de twee bomen met het kleinste gewicht samen. Het gewicht van de nieuwe boom is de som van de gewichten van de twee subbomen.
- Herhaal de vorige stap totdat er nog maar 1 boom is.

THRijswijk

Compression

Implementatie

- ibstream
  - zorgt voor bitgewijze input.
- obstream
  - zorgt voor bitgewijze output.
  - flush moet private zijn.
- CharCounter
  - een frequentietabel
  - bevat map<char, int>
- HuffmanTree
  - de coderings trie
  - gebruikt priority_queue om trie te bouwen.
  - gebruik deque<int> om variable length codes op te slaan.
- Compressor
  - compression en uncompression.

THRijswijk

HuffmanTree

```
class HuffmanTree {
public:
   HuffmanTree();
   HuffmanTree(const CharCounter& cc);
   enum {ERROR=-3,
         INCOMPLETE_CODE=- 2, END=-1};
   deque<int> getCode(int ch);
   int getChar(const deque<int>& code) const;
   void writeEncodingTable(ostream& out);
   void readEncodingTable(istream& in);
private:
   class N {
   public:
      N* left; N* right; N* parent;
      int value; int weight;
      N(int v, int w, N* lt, N* rt, N* pt);
   };
   CharCounter theCounts;
   map<int, N*> theNodes;
   N* root;
   void createTree();
   deque<int> getCode(N* current);
friend bool greaterWeight(N* lhs, N* rhs);
};
```

THRijswijk

HuffmanTree

```
HuffmanTree::HuffmanTree(
         const CharCounter& cc):
      theCounts(cc), root(0) {
   createTree();
}

bool greaterWeight(HuffmanTree::N* lhs,
      HuffmanTree::N* rhs) {
   return lhs->weight>rhs->weight;
}
```

THRijswijk

createTree

```
void HuffmanTree::createTree() {
   priority_queue<N*, deque<N*>,
      bool (*)(N* lhs, N* rhs)> pq(greaterWeight);
   for (int i(0); i<DIFF_CHARS; ++i) {
      char ci(static_cast<char>(i));
      if (theCounts.getCount(ci)>0) {
         N* newNode(new N(
            i, theCounts.getCount(ci), 0, 0, 0));
         theNodes[i]=newNode;
         pq.push(newNode);
      }
   }
   theNodes[END]=new N(END, 1, 0, 0, 0);
   pq.push(theNodes[END]);
   while (pq.size()>1) {
      N* n1(pq.top()); pq.pop();
      N* n2(pq.top()); pq.pop();
      N* result(new N(INCOMPLETE_CODE,
         n1- >weight+n2->weight, n1, n2, 0));
      n1->parent=n2->parent=result;
      pq.push(result);
   }
   root=pq.top();
}
```

THRijswijk

getCode

```
deque<int> HuffmanTree::getCode(int ch) {
   if (root==0)
      return deque<int>();
   return getCode(theNodes[ch]);
}

deque<int> HuffmanTree::getCode(N* current) {
   deque<int> v;
   N* par(current->parent);
   while (par != 0) {
      if(par->left==current)
         v.push_front(0);
      else
         v.push_front(1);
      current=current->parent;
      par=current->parent;
   }
   return v;
}
```

THRijswijk

getChar

```
int HuffmanTree::getChar(
      const deque<int>& code) const {
   N* p(root);
   for (deque<int>::size_type i(0);
         p!=0 && i<code.size(); ++i)
      if (code[i]==0)
         p=p->left;
      else
         p=p->right;
   if (p==0)
      return ERROR;
   return p->value;
}
```

THRijswijk