Laporan Tugas Kecil I Strategi Algoritma

Penyelesaian Permainan Kartu 24 dengan Algoritma *Brute Force*

Logo

Description automatically generated

Disusun oleh

Nama: Haziq Abiyyu Mahdy

Kelas: K2

NIM: 13521170

Tahun ajaran 2022/2023

**BAB I**

**DESKRIPSI PERSOALAN**

Kartu 24 adalah salah satu variasi permainan kartu yang dapat dimainkan dengan kartu remi. Kartu remi terdiri dari lima puluh dua kartu yang terbagi menjadi empat suit (sekop, hati, keriting, dan wajik) yang masing-masing terdiri dari tiga belas kartu (As, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, dan King). As bernilai 1, Jack bernilai 11, Queen bernilai 12, King bernilai 13, sedangkan kartu bilangan memiliki nilai dari bilangan itu sendiri. Mekanisme permainan kartu 24 adalah sebagai berikut. Pada awal permainan, moderator atau salah satu pemain mengambil empat kartu dari dek yang sudah dikocok. Kemudian pemain akan menebak susunan serta operasi aritmatika yang dapat menghasilkan angka dua puluh empat. Operasi yang diperbolehkan adalah penjumlahan (+), pengurangan (-), perkalian (\*), pembagian (/), dan tanda kurung (()). Tiap kartu harus digunakan tepat sekali dan urutan penggunaannya bebas.

Pada tugas ini penulis merancang program dalam bahasa C++ untuk menerima empat input berupa angka atau huruf yang merupakan bagian dari dari (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K) atau *generate* secara *random*, kemudian menampilkan output berupa banyaknya solusi yang ditemukan, solusi dari permainan yang kemudian dapat disimpan ke dalam *file*, dan waktu eksekusi yang diperlukan. Algoritma yang digunakan untuk mendapatkan solusi permainan kartu 24 adalah algoritma *brute force*.

**BAB II**

**ALGORITMA PENYELESAIAN MASALAH**

Pada bagian awal program, program akan menanyakan apakah pengguna ingin melakukan input kartu atau mendapatkan kartu *random*. Jika pengguna ingin melakukan input kartu, maka pengguna dapat mengetikkan empat angka atau huruf yang terdiri dari (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K) yang dibatasi oleh spasi seperti contoh berikut.

|  |
| --- |
| Insert card numbers:  A 3 Q K |

Kemudian, program akan melakukan *parsing* dan validasi pada masukan pengguna. Jika masukan tidak valid, maka akan muncul pesan kesalahan dan pengguna diminta melakukan input ulang hingga program valid. Jika masukan valid, maka program akan menyimpan vector<int> berupa nilai dari keempat kartu yang diinput secara terurut. Jika pengguna ingin mendapatkan kartu secara *random*, maka program akan memilih kartu secara random, menampilkan kartu di layar dan menyimpan nilai kartu tersebut ke dalam vector<int>.

|  |
| --- |
| Your cards:  3 3 J A |

Kemudian, program akan mencari seluruh susunan/permutasi kartu yang mungkin diperoleh dari input kartu yang ada. Pencarian permutasi kartu dilakukan dengan menggunakan prosedur void findNumberPermutation(vector<int>& container, int leftIdx, int rightIdx). Berikut adalah penjelasan algoritma yang digunakan pada prosedur tersebut. Pada pemanggilan prosedur, kita memasukkan parameter berupa vector<int> yang sudah diperoleh dari proses input sebelumnya, leftIdx = 0, dan rightIdx = TOTALNUMS – 1 (=3). Misalkan vector<int> yang diperoleh adalah {2, 3, 3, 4}. Maka, *initial state* prosedur dapat dinyatakan pada gambar berikut.

Table

Description automatically generated

Program akan mencari elemen yang akan dijadikan *prefix*, yaitu elemen yang sudah tetap posisinya dan tidak akan ditukar lagi, dengan cara melakukan iterasi dari i = leftIdx → i = rightIdx dan *swap* Vec[leftIdx] dengan Vec[i]. Setelah itu, elemen Vec[0 .. leftIdx] akan dijadikan *prefix* dan elemen Vec[leftIdx+1 .. rightIdx] akan dicari permutasinya, lalu dilakukan kembali swap Vec[leftIdx] dengan Vec[i] untuk backtracking.

Namun, tidak semua elemen dari Vec[leftIdx .. rightIdx] boleh di-*swap* dengan Vec[leftIdx]. Jika pada Vec[leftIdx .. i-1] terdapat elemen yang sama dengan Vec[i], dan Vec[leftIdx] di*swap* dengan Vec[i], maka akan menghasilkan prefix yang sama dengan hasil *swap* yang sudah dilakukan sebelumnya. Berikut adalah ilustrasinya.

Diagram

Description automatically generated

Pada saat i = 1, proses *swap* Vec[leftIdx] dengan Vec[i] akan menghasilkan [3, perm([2, 3, 4])]. Saat i = 2, jika dilakukan *swap* Vec[leftIdx] dengan Vec[i], maka akan dihasilkan [3, perm([3, 2, 4])]. Karena kedua vector tersebut memiliki *prefix* yang sama, yaitu 3, dan seluruh susunan yang dihasilkan oleh perm([3, 2, 4]) juga akan dihasilkan oleh perm([2, 3 4]), maka tidak perlu dilakukan *swap* pada i = 2 agar tidak menghasilkan duplikat pada hasil permutasi. Rekursi terus dilakukan hingga leftIdx = rightIdx, atau tidak ada lagi elemen yang dapat di-*swap* (kasus basis) seperti pada gambar berikut.

Chart, box and whisker chart

Description automatically generated

Setiap rekursi mencapai kasus basis, program akan menyimpan vector permutasi angka dengan melakukan push pada variabel global vector<vector<int>> numsPermutation.

Setelah itu, program akan mencari seluruh susunan/permutasi operasi yang dapat dilakukan dengan menggunakan prosedur void findOpsPermutation(). Karena setiap operasi yang tersedia (+, -, \*, /) unik, kita tidak perlu mengkhawatirkan adanya duplikat atau permutasi yang muncul lebih dari satu kali. Oleh karena itu, kita dapat menggunakan *nested for* untuk memperoleh susunan operasi pertama, kedua, dan ketiga. Setiap permutasi akan dimasukkan/di-*push* ke dalam vector<vector<char>>.

Diagram

Description automatically generated

Setelah memperoleh seluruh permutasi angka dan operasi, kita dapat melakukan iterasi pada vector susunan operasi dan vector susunan angka dan melakukan evaluasi pada ekspresi yang diperoleh.

|  |
| --- |
| for (int i=0; i < opsPermutations.size(); i++) {  for (int j=0; j < numsPermutations.size(); j++) {  /\* Lakukan evaluasi pada ekspresi yang ada pada susunan opsPermutations[i] dan numsPermutations[j] \*/  }  } |

Untuk peletakan tanda kurung, terdapat lima kasus yang dapat dieksekusi secara manual, yaitu;

|  |
| --- |
| (a op1 b) op2 (c op3 d)  a op1 (b op2 (c op3 d))  a op1 ((b op2 c) op3 d)  ((a op1 b) op2 c) op3 d  (a op1 (b op2 c)) op3 d |

Evaluasi ekspresi akan dilakukan pada setiap kasus peletakan tanda kurung, dan jika evaluasi menghasilkan angka 24, maka jumlah solusi akan di-*increment* dan string ekspresi akan di-*push* ke dalam vector<string> savedResults.

**BAB III**

**SOURCE CODE**

Program terdiri atas satu *header file* (header.hpp) dan tiga modul (iofile.cpp untuk implementasi fungsi yang berkaitan dengan I/O, algorithm.cpp untuk implementasi fungsi yang berkaitan dengan algoritma, dan driver.cpp)

1. header.hpp

|  |
| --- |
| #ifndef HEADER\_HPP  #define HEADER\_HPP  #include <iostream>  #include <vector>  #include <fstream>  #include <chrono>  using namespace std;  // CONSTANTS  #define TOTALNUMS 4  #define TOTALOPS 4  // GLOBAL VARIABLES  extern vector<vector<int>> numsPermutations;  extern vector<vector<char>> opsPermutations;  extern vector<string> savedResults;  // IO RELATED FUNCTIONS  vector<string> inputCardFromUser();  vector<int> generateRandomCard();  vector<int> getNumsVec();  /\* Gets input from user, validates the input, and converts the input into vector<int> \*/  bool isInputValid(vector<string> inputs);  bool isInputElmtValid(string input);  int convertToInt(string input);  /\* Precondition: input string is already valid \*/  void printResults();  void saveResultsToFile(int solutions);  // ALGORITHM  float evaluate(float firstNum, float secondNum, char op);  void swapElmt(int& num1, int& num2);  void findNumberPermutation(vector<int>& container, int leftIdx, int rightIdx);  void findOpsPermutation();  int getPossibleResults();  #endif |

1. iofile.cpp

|  |
| --- |
| #include "header.hpp"  vector<string> inputCardFromUser() {  vector<string> inputs;  string buffer;  int currentSize = 0;  cout << "Insert card numbers: " << endl;  getline(cin, buffer);  if (buffer == "") {  getline(cin, buffer);  }  string currentWord = "";  for (int i=0; i <= buffer.length(); i++) {  if (buffer[i] == ' ' || !buffer[i]) {  if (currentWord != "") {  inputs.push\_back(currentWord);  currentSize++;  currentWord = "";  if (currentSize == 5) break;  // Kalau size nya udah 5 langsung break aja karena udah pasti ga valid  }  }  else {  currentWord += buffer[i];  }  }  return inputs;  }  vector<int> generateRandomCard() {  const vector<string> cardOptions = {"A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"};  vector<int> randomCard(TOTALNUMS);  int random;  srand((unsigned) time(NULL));  cout << "Your cards: " << endl;  for (int i=0; i < TOTALNUMS; i++) {  random = rand() % 13;  cout << cardOptions[random] << " ";  randomCard[i] = convertToInt(cardOptions[random]);  }  cout << endl << endl;  return randomCard;  }  vector<int> getNumsVec() {  vector<string> temp;  vector<int> numsVec(TOTALNUMS);  temp = inputCardFromUser();  while (!isInputValid(temp)) {  cout << "Invalid input!" << endl;  temp = inputCardFromUser();  }  cout << endl;    for (int i=0; i < TOTALNUMS; i++) {  numsVec[i] = convertToInt(temp[i]);  }    return numsVec;  }  bool isInputValid(vector<string> inputs) {  if (inputs.size() != TOTALNUMS) {  return false;  }  for (int i=0; i < TOTALNUMS; i++) {  if (!isInputElmtValid(inputs[i])) {  return false;  }  }  return true;  }  bool isInputElmtValid(string input) {  if (input.length() == 2) {  return (input == "10");  }  if (input.length() == 1) {  return ((input[0] - '0' > 1 && input[0] - '0' < 10) ||  (input[0] == 'A') ||  (input[0] == 'J') ||  (input[0] == 'Q') ||  (input[0] == 'K'));  }  else {  return false;  }  }  int convertToInt(string input) {  if (input == "10") {return 10;}  else if (input[0] - '0' > 1 && input[0] - '0' < 10) {return input[0] - '0';}  else if (input[0] == 'A') {return 1;}  else if (input[0] == 'J') {return 11;}  else if (input[0] == 'Q') {return 12;}  else /\* (input[0] == 'K') \*/ {return 13;}  }  void printResults() {  for (int i=0; i < savedResults.size(); i++) {  cout << savedResults[i] << endl;  }  cout << endl;  }  void saveResultsToFile(int solutions) {  string fileName;  ofstream fout;  cout << "Insert file name: ";  cin >> fileName;  fout.open("test/" + fileName);  if (!solutions) {  fout << "No solutions found" << endl;  }  else {  fout << solutions << " solutions found" << endl;  for (int i=0; i < savedResults.size(); i++) {  fout << savedResults[i] << endl;  }  }  } |

1. algorithm.cpp

|  |
| --- |
| #include "header.hpp"  vector<vector<int>> numsPermutations;  vector<vector<char>> opsPermutations;  vector<string> savedResults;  float evaluate(float firstNum, float secondNum, char op) {  if (op == '+') {return firstNum + secondNum;}  else if (op == '-') {return firstNum - secondNum;}  else if (op == '\*') {return firstNum \* secondNum;}  else /\* (op == '/') \*/ {return firstNum / secondNum;}  }  void swapElmt(int& num1, int& num2) {  int temp = num1;  num1 = num2;  num2 = temp;  }  bool shouldSwap(vector<int>& container, int leftIdx, int currentIdx) {  for (int i=leftIdx; i < currentIdx; i++) {  if (container[i] == container[currentIdx]) {  return false;  }  }  return true;  }  void findNumberPermutation(vector<int>& container, int leftIdx, int rightIdx) {  if (leftIdx == rightIdx) {  numsPermutations.push\_back(container);  return;  }  for (int i=leftIdx; i <= rightIdx; i++) {  bool check = shouldSwap(container, leftIdx, i);  if (check) {  swapElmt(container[leftIdx], container[i]);  findNumberPermutation(container, leftIdx + 1, rightIdx);  swapElmt(container[leftIdx], container[i]);  }  }  }  void findOpsPermutation() {  vector<char> temp(3);  const char allOps[TOTALOPS] = {'+', '-', '\*', '/'};  for (int i=0; i < TOTALOPS; i++) {  for (int j=0; j < TOTALOPS; j++) {  for (int k=0; k < TOTALOPS; k++) {  temp[0] = allOps[i];  temp[1] = allOps[j];  temp[2] = allOps[k];  opsPermutations.push\_back(temp);  }  }  }  }  int getPossibleResults() {  int solutions = 0;  int a, b, c, d;  float result;  char op1, op2, op3;  string temp;  for (int i=0; i < opsPermutations.size(); i++) {  for (int j=0; j < numsPermutations.size(); j++) {  a = numsPermutations[j][0];  b = numsPermutations[j][1];  c = numsPermutations[j][2];  d = numsPermutations[j][3];  op1 = opsPermutations[i][0];  op2 = opsPermutations[i][1];  op3 = opsPermutations[i][2];  // (a op1 b) op2 (c op3 d)  result = evaluate(evaluate(a, b, op1), evaluate(c, d, op3), op2);  if (result == 24) {  temp = "(" + to\_string(a) + op1 + to\_string(b) + ")" + op2 + "(" + to\_string(c) + op3 + to\_string(d) + ")";  savedResults.push\_back(temp);  solutions++;  }  // a op1 (b op2 (c op3 d))  result = evaluate(a, evaluate(b, evaluate(c, d, op3), op2), op1);  if (result == 24) {  temp = to\_string(a) + op1 + "(" + to\_string(b) + op2 + "(" + to\_string(c) + op3 + to\_string(d) + "))";  savedResults.push\_back(temp);  solutions++;  }  // a op1 ((b op2 c) op3 d)  result = evaluate(a, evaluate(evaluate(b, c, op2), d, op3), op1);  if (result == 24) {  temp = to\_string(a) + op1 + "((" + to\_string(b) + op2 + to\_string(c) + ")" + op3 + to\_string(d) + ")";  savedResults.push\_back(temp);  solutions++;  }  // ((a op1 b) op2 c) op3 d  result = evaluate(evaluate(evaluate(a, b, op1), c, op2), d, op3);  if (result == 24) {  temp = "((" + to\_string(a) + op1 + to\_string(b) + ")" + op2 + to\_string(c) + ")" + op3 + to\_string(d);  savedResults.push\_back(temp);  solutions++;  }  // (a op1 (b op2 c)) op3 d  result = evaluate(evaluate(a, evaluate(b, c, op2), op1), d, op3);  if (result == 24) {  temp = "(" + to\_string(a) + op1 + "(" + to\_string(b) + op2 + to\_string(c) + "))" + op3 + to\_string(d);  savedResults.push\_back(temp);  solutions++;  }  }  }  return solutions;  } |

1. driver.cpp

|  |
| --- |
| #include "header.hpp"  int main() {  vector<int> test;  int solutions;  string option;    while (true) {  cout << "Do you want to insert cards (0) or generate random cards (1)?: ";  cin >> option;  if (option == "0") {  test = getNumsVec();  break;  }  if (option == "1") {  test = generateRandomCard();  break;  }  cout << "Invalid input!" << endl;  }    using namespace std::chrono;  high\_resolution\_clock::time\_point start = high\_resolution\_clock::now();  findNumberPermutation(test, 0, TOTALNUMS-1);  findOpsPermutation();  solutions = getPossibleResults();  high\_resolution\_clock::time\_point end = high\_resolution\_clock::now();  duration<double> executionTime = duration\_cast<duration<double>>(end - start);  if (!solutions) {  cout << "No solutions found" << endl;  }  else {  cout << solutions << " solutions found" << endl;  printResults();  }  cout << "execution time: " << executionTime.count() \* 1000 << " ms" << endl << endl;  while (true) {  cout << "Do you want to save the result? (Y/N): ";  cin >> option;  if (option == "Y") {  saveResultsToFile(solutions);  break;  }  if (option == "N") {  break;  }  cout << "Invalid input" << endl;  }    return 0;  } |

**BAB IV**

**KASUS UJI DAN *CHECKLIST***

1. Kasus uji pertama

Text

Description automatically generated

1. Kasus uji kedua

Text

Description automatically generated

Text

Description automatically generated

(P.S. Karena terdapat sangat banyak solusi, tidak semua solusi dapat ditampilkan pada *screenshot* ini. Untuk selengkapnya dapat dilihat di test/testcase2.txt)

1. Kasus uji ketiga

Text

Description automatically generated

1. Kasus uji keempat

Text

Description automatically generated

1. Kasus uji kelima

Text

Description automatically generated

1. Kasus uji keenam

Text

Description automatically generated

Table

Description automatically generated

**BAB V**

**LINK TO REPOSITORY**

*Repository* dapat diakses melalui tautan berikut:

<https://github.com/haziqam/Tucil1_13521170>