**Laporan Tugas Kecil 2**

**Pengaturan Jadwal Kelas Dengan *Topological Sort***

Shifa Salsabiila 13519106

Program Studi Teknik Informatika

IF2211: Strategi Algoritma

Dr. Nur Ulfa Maulidevi, S.T., M.Sc.

# **Daftar Isi**

[Daftar Isi ii](#_Toc62619627)

[I. Algoritma 1](#_Toc62619628)

[II. Source Program 3](#_Toc62619629)

[III. Hasil dan Test Case 6](#_Toc62619630)

[IV. Alamat Source Code 11](#_Toc62619631)

# **Algoritma**



**Gambar 1.1** Diagram alur program

Program diawali dengan membaca sebuah file teks yang berisi *n* baris, dengan ketentuan: *n-2* baris pertama merupakan *operand*, dan *operand* terakhir diakhiri dengan tanda tambah (+). Baris berikutnya merupakan pemisah antara *operand* dan hasil. Baris terakhir adalah hasil penjumlahan. Setelah dibaca, seluruh *operand* akan disimpan dalam sebuah *array* dan hasil akan disimpan dalam sebuah variabel. Kemudian, akan dibuat sebuah *array* baru, yang menyimpan seluruh karakter yang ada pada gabungan *operand* dan hasil tanpa pengulangan, selanjutnya akan direferensikan sebagai *dictionary*. Selanjutnya, dibuat juga sebuah array yang berisi angka 0 – 9 (inklusif), yang direferensikan sebagai *nums*.

Bagian utama algoritma *brute force* yang digunakan mencakup proses-proses berikut: kombinasi, permutasi, pengubahan kata-kata menjadi bilangan, pengecekan validitas hasil, dan pengecekan kebenaran ekspresi. Pertama, akan dilakukan kombinasi dari *nums.* Akan diperoleh sebanyak C(n, r) hasil, dengan n: panjang *array nums* dan r: panjang *array dictionary*. Kemudian, masing-masing solusi kombinasi akan dimasukkan pada fungsi permutasi. Masing-masing input akan menghasilkan sebanyak r! hasil permutasi. Berikutnya, masing-masing hasil permutasi akan menjadi basis untuk perumusan transformasi kata-kata pada input menjadi sebuah bilangan.

Selanjutnya, hasil transformasi akan dicek apakah memenuhi syarat bahwa, tidak ada huruf pertama yang merupakan 0. Jika memenuhi, kemudian akan dilakukan pengecekan apakah ketika ekspresi penjumlahan *operand* yang sudah ditransformasi sama dengan hasil yang juga sudah ditransformasi. Jika ya, maka jawaban akan disimpan. Proses utama ini akan diulang hingga tidak ada lagi kemungkinan jawaban lain.

Pada proses ini, perlu diperhatikan bahwa pemanggilan fungsi permutasi, pengubahan kata-kata menjadi bilangan, pengecekan validitas hasil, dan pengecekan kebenaran ekspresi dilakukan di dalam fungsi kombinasi, sehingga program tidak perlu melakukan dua kali proses pengulangan.

# ***Source Program***

Bahasa: C++

**ADT Graph**

|  |
| --- |
| #ifndef **GRAPH\_HPP**  #define **GRAPH\_HPP**  #include <iostream>  #include <string>  using namespace **std**;  typedef struct **tadrNode** \***adrNode**;  typedef struct **tadrSuccNode** \***adrSuccNode**;  typedef struct **tadrNode** {  int NPred;  **string** Id;  **adrNode** Next;  **adrSuccNode** Trail;  } **Node**;  typedef struct **tadrSuccNode** {  **adrNode** Succ;  **adrSuccNode** NextT;  } **SuccNode**;  typedef struct {  **adrNode** First;  } **Graph**;  #define **First**(G) (G).First  #define **Id**(P) (P)->Id  #define **NPred**(P) (P)->NPred  #define **Trail**(P) (P)->Trail  #define **Succ**(Pt) (Pt)->Succ  #define **NextT**(Pt) (Pt)->NextT  #define **Next**(P) (P)->Next  **adrNode** **AlokNode**(**string** X);  void **DealokNode**(**adrNode** P);  **adrSuccNode** **AlokSuccNode**(**adrNode** Pn);  void **DealokSuccNode**(**adrSuccNode** Pt);  void **CreateGraph** (**Graph** \*G, **string** X);  **adrNode** **SearchNode**(**Graph** G, **string** X);  **adrSuccNode** **SearchEdge**(**Graph** G, **string** prec, **string** succ);  void **InsertNode**(**Graph** \*G, **string** X, **adrNode** \*Pn);  void **DeleteNode**(**Graph** \*G, **string** X);  void **InsertEdge**(**Graph** \*G, **string** prec, **string** succ);  void **DeleteAllEdges**(**Graph** \*G, **adrNode** Pn);  #endif |

|  |
| --- |
| **adrNode** **AlokNode**(**string** X) {  */\*KAMUS\*/*  **Node** \*P = (**Node**\*) **malloc**(sizeof(**Node**));  */\*ALGORITMA\*/*  if (P != **NULL**) {  **Id**(P) **=** X;  **NPred**(P) = 0;  **Trail**(P) = **NULL**;  **Next**(P) = **NULL**;  }  return P;  } |

|  |
| --- |
| void **DealokNode**(**adrNode** P) {  */\*KAMUS\*/*  */\*ALGORITMA\*/*  **free**(P);  } |

|  |
| --- |
| **adrSuccNode** **AlokSuccNode**(**adrNode** Pn) {  */\*KAMUS\*/*  **adrSuccNode** Pt;  */\*ALGORITMA\*/*  Pt = (**SuccNode**\*) **malloc** (sizeof(**SuccNode**));  if (Pt != **NULL**) {  **Succ**(Pt) = Pn;  **NextT**(Pt) = **NULL**;  }  return Pt;  } |

|  |
| --- |
| void **DealokSuccNode**(**adrSuccNode** Pt) {  */\*KAMUS\*/*  */\*ALGORITMA\*/*  **free**(Pt);  } |

|  |
| --- |
| **adrNode** **SearchNode**(**Graph** G, **string** X) {  */\*KAMUS\*/*  **adrNode** P;  */\*ALGORITMA\*/*  P = **First**(G);  while (P != **NULL** && **Id**(P) **!=** X){  P = **Next**(P);  }  return P;  } |

|  |
| --- |
| **adrSuccNode** **SearchEdge**(**Graph** G, **string** prec, **string** succ) {  */\*KAMUS\*/*  **adrNode** P;  **adrSuccNode** T;  */\*ALGORITMA\*/*  P = **SearchNode**(G, prec);    if (P == **NULL**){  return **NULL**;  } else{  T = **Trail**(P);  if (T == **NULL**){  return **NULL**;  } else{  while ((**Id**(**Succ**(T)) **!=** succ) && (**NextT**(T) != **NULL**)) {  T = **NextT**(T);  }  if (**NextT**(T) == **NULL**){  return **NULL**;  } else{  return T;  }  }  }  } |

|  |
| --- |
| void **DeleteNode**(**Graph** \*G, **string** X) {  */\*KAMUS\*/*  **adrNode** P;  **adrNode** Prev;  */\*ALGORITMA\*/*  P = **SearchNode**(\*G, X);  Prev = **First**(\*G);  *//Removing from main list*  if (P != **NULL**) {  if (P == **First**(\*G)) {  **First**(\*G) = **Next**(P);  } else {  while (**Next**(Prev)!=P) {  Prev = **Next**(Prev);  }  **Next**(Prev) = **Next**(P);  }  *//Removing all edges that P is connected to*  **DeleteAllEdges**(G, P);  }  } |

|  |
| --- |
| void **InsertEdge**(**Graph** \*G, **string** prec, **string** succ) {  */\*KAMUS\*/*  **adrNode** Pprec;  **adrNode** Psucc;  **adrSuccNode** T;  */\*ALGORITMA\*/*  Pprec = **SearchNode**(\*G,prec);  Psucc = **SearchNode**(\*G,succ);  if (**SearchEdge**(\*G,prec,succ) == **NULL**){  T = **Trail**(Pprec);  if (T == **NULL**){  **Trail**(Pprec) = **AlokSuccNode**(Psucc);  }  else{  while (**NextT**(T) != **NULL**){  T = **NextT**(T);  }  **NextT**(T) = **AlokSuccNode**(Psucc);    }  **NPred**(Psucc)++;  }  } |

|  |
| --- |
| void **DeleteAllEdges**(**Graph** \*G, **adrNode** Pn) {  */\*KAMUS\*/*  **adrSuccNode** Pt;  */\*ALGORITMA\*/*  Pt = **Trail**(Pn);  while (Pt != **NULL**) {  **NPred**(**Succ**(Pt)) -= 1;  Pt = **NextT**(Pt);  }  } |

**Pembacaan File**

|  |
| --- |
| void **readFile**(**string** path, **Graph** \*G) {  */\*KAMUS\*/*  **ifstream** fileInput;  **string** x;  **string**\* array;  int lines\_read;  */\*ALGORITMA\*/*  lines\_read = 0;  fileInput.**open**(path);  array = new **string**[100];  if (**!**fileInput) {  cerr **<<** "File not found!" **<<** **endl**;  }  while (**getline**(fileInput, x)) {  **readLine**(array, x, G, lines\_read);  lines\_read += 1;  }  } |

|  |
| --- |
| void **readLine**(**string**\* array, **string** line, **Graph** \*G, int lines\_read) {  */\*KAMUS\*/*  **string** delims = ",. ";  **size\_t** pos = 0;  **string** token;  int i = 0;  */\*ALGORITMA\*/*  while ((pos = line.**find\_first\_of**(delims)) != **string**::npos) {  token **=** line.**substr**(0, pos);  if (token**!=**"") {  array[i] **=** token;  i += 1;  }  line.**erase**(0, pos + 1);  }  **addLineToGraph**(array, i, G, lines\_read);  } |

**Pembuatan Graph**

|  |
| --- |
| void **addLineToGraph**(**string**\* array, int n, **Graph** \*G, int lines\_read) {  */\*KAMUS\*/*  **adrNode** P;  **string** x = array[0];  */\*ALGORITMA\*/*  *//First node*  if (lines\_read == 0) {  **CreateGraph**(G, x);  } else {  if (**SearchNode**(\*G, array[0]) == **NULL**) {  **InsertNode**(G, array[0], &P);  }  }  *//cout << "First node is: " << Id(First(\*G)) << endl;*  for (int i=1; i<n; i++) {  if (**SearchNode**(\*G, array[i]) == **NULL**) {  **InsertNode**(G, array[i], &P);  }  **InsertEdge**(G, array[i], array[0]);  }  } |

**Topological Sort**

|  |
| --- |
| void **generateJadwal**(**Graph** \*G, int sem) {  */\*KAMUS\*/*  **adrNode** P;  **string**\* toDelete;  int i, j;  */\*ALGORITMA\*/*  i = 0, j = 0;  P = **First**(\*G);  toDelete = new **string**[100];  cout **<<** "Semester " **<<** sem **<<** ": " ;  while (P != **NULL**) {  if (**NPred**(P)==0) {  cout **<<** **Id**(P) **<<** " ";  toDelete[i] **=** **Id**(P);  i += 1;  }  P = **Next**(P);  }  cout **<<** **endl**;  *//Deleting all nodes*  while (j<i) {  **DeleteNode**(G, toDelete[j]);  j += 1;  }  delete[] toDelete;  } |

**Main Program**

|  |
| --- |
| int **main** () {  */\*KAMUS\*/*  **Graph** G;  int sem;  */\*ALGORITMA\*/*  **readFile**("../test/{namafile}.txt", &G);  sem = 1;  while (**First**(G) != **NULL**) {  **generateJadwal**(&G, sem);  sem += 1;  }    return 0;  } |

# **Hasil dan *Test Case***



**Gambar 3.1** Prompt input(?)



**Gambar 3.2** Tampilan pembacaan nama file

**Tabel 3.1** *Test case*

|  |  |
| --- | --- |
| **Input** | **Output** |
| File: test.txt | **Runtime: 0.87798**  **Checks: 105840** |

**Tabel 3.2** Form penilaian mandiri

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Poin** | **Ya** | **Tidak** |
| 1. | Program berhasil dikompilasi. | √ |  |
| 2. | Program berhasil *running.* | √ |  |
| 3. | Program dapat menerima berkas input dan menuliskan output. | √ |  |
| 4. | Luaran sudah benar untuk semua kasus input. | √ |  |

# **Alamat *Source Code***

Repository: <https://github.com/salsabiilashifa11/TopSort>