TUGAS AKHIR - KI141502

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI GRAF HAMILTON PADA PENYELESAIAN PERSOALAN: SPOJ KLASIK 8750 WORD PLAY

PRAMUDITO HAPRIARSO
NRP 05111440000035

Dosen Pembimbing 1
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2
Dwi Sunaryono, S.Kom., M.Kom.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

Halaman ini sengaja dikosongkan

TUGAS AKHIR - KI141502

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI GRAF HAMILTON PADA PENYELESAIAN PERSOALAN: SPOJ KLASIK 8750 WORD PLAY

PRAMUDITO HAPRIARSO
NRP 05111440000035

Dosen Pembimbing 1
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2
Dwi Sunaryono, S.Kom., M.Kom.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

Halaman ini sengaja dikosongkan

UNDERGRADUATE THESES - KI141502

DESIGNS AND ANALYTICS OF HAMILTONIAN GRAPH COMPUTATIONAL ALGORITHM ON CLASSICAL PROBLEM SOLUTION SPOJ 8750 WORD PLAY

PRAMUDITO HAPRIARSO
NRP 05111440000035

Supervisor 1
Rully Soelaiman, S.Kom., M.Kom.

Supervisor 2
Dwi Sunaryono, S.Kom., M.T.

INFORMATICS DEPARTMENT
Faculty of Information and Communication Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

Halaman ini sengaja dikosongkan

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI GRAF HAMILTON PADA PENYELESAIAN PERSOALAN: SPOJ KLASIK 8750 WORD PLAY

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :
Pramudito Hapriarso
NRP : 05111440000035

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Rully Soelaiman, S.Kom., M.Kom. ..... (Pembimbing 1)
NIP: 19700213 199402 1 001

Dwi Sunaryono, S.Kom., M.Kom. ..... (Pembimbing 2)
NIP: 19720528 199702 1 001

SURABAYA
JULI 2018

Halaman ini sengaja dikosongkan

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI GRAF HAMILTON PADA PENYELESAIAN PERSOALAN: SPOJ KLASIK 8750 WORD PLAY

Nama	: PRAMUDITO HAPRIARSO
NRP	: 05111440000035
Departemen	: Informatika FTIK-ITS
Pembimbing 1	: Rully Soelaiman, S.Kom., M.Kom.
Pembimbing 2	: Dwi Sunaryono, S.Kom., M.Kom.

Abstrak

Permasalahan yang dibahas adalah permasalahan penyusunan beberapa buah string sepanjang K menjadi sebuah string sepanjang N yang diinginkan. Diberikan sebuah kata dengan panjang N yang dipisahkan menjadi N-K+1 buah subkata dengan panjang K dan diurutkan dengan urutan leksikografis. Diperlukan sebuah solusi unik untuk menyusun kembali subkata-subkata yang telah urut berdasarkan urutan leksikografis tadi menjadi sebuah kata utuh yang diinginkan.

Penyelesaian permasalahan dilakukan dengan cara merepresentasikan masukan permasalahan tersebut ke dalam bentuk graf. Dengan menjadikan subkata-subkata yang diberikan sebagai simpul (node) dari sebuah graf. Graf yang akan dibentuk merupakan graf berarah yang mengandung lintasan Hamilton. Sebuah graf berarah dapat dikatakan memiliki lintasan hamilton apabila masing-masing simpul (node) dari graf tersebut dapat dilewati sejumlah tepat satu kali. Sebelum digunakan, subkata-subkata yang menjadi masukan sistem ditampung ke dalam container map. Container

map ini telah disediakan oleh Standard Template Library (STL). Selain menyediakan container, Standard Template Library (STL) ini juga menyediakan komponen-komponen lain seperti algoritma, fungsi, dan juga iterators yang dapat digunakan.

Pada tugas akhir ini akan dirancang penyelesaian masalah yang disampaikan pada paragraf pertama dengan menggunakan pengimplementasian metode graf hamilton dan juga penggunaan Standard Template Library (STL). Solusi yang dikembangkan berjalan dengan kompleksitas waktu $O((N-K+1)\log_2(N-K+1))$*, dimana N adalah panjang kata yang dicari, K adalah panjang subkata yang menjadi masukan.*

Kata Kunci: graf, graf hamilton, SPOJ word play, string

DESIGNS AND ANALYTICS OF HAMILTONIAN GRAPH COMPUTATIONAL ALGORITHM ON CLASSICAL PROBLEM SOLUTION SPOJ 8750 WORD PLAY

Name	: PRAMUDITO HAPRIARSO
NRP	: 05111440000035
Major	: Informatics Department Faculty of IC-ITS
Supervisor 1	: Rully Soelaiman, S.Kom., M.Kom.
Supervisor 2	: Dwi Sunaryono, S.Kom., M.Kom.

Abstract

Problem discussed is a constructing some K-length string problem to become an expected N-length string. Given a word with length N divided into as many as N-K+1 subwords with length K and lexicographically sorted. An unique solution is needed to arrange these lexicographically sorted subwords into an expected word.

The problem's solution is ilustrating the problem input into graph. Assuming these subwords given as a graph edges (nodes). Graph to be formed is a directed graph having hamiltonian path. A directed graph regarded as a graph having hamiltonian path if each edge (node) can be visited exactly once. Substrings that become the system input is collected and saved into map container before it is used. This map container is provided by Standard Template Library (STL). Besides providing container, Standard Template Library (STL) also provides other components like algorithm, function and iterators that can be used.

In this final project, the writer will design and analyze an algorithm to solve the problem that stated in the first paragraph using implementation of hamiltonian graph method and also using Standard Template Library (STL). Developed solutions working with time complexity $O((N-K+1)\log_2(N-K+1))$*, where N is length of expected word, K is length of input substrings.*

Keywords: graph, hamiltonian graph, SPOJ word play, string

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Allah SWT atas rezeki, berkah, kekuatan dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul :

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI GRAF HAMILTON PADA PENYELESAIAN PERSOALAN: SPOJ KLASIK 8750 WORD PLAY

Dengan selesainya Tugas Akhir ini diharapkan apa yang telah dikerjakan penulis dapat memberikan kontribusi bagi perkembangan ilmu pengetahuan terutama di bidang teknologi informasi serta bagi diri penulis sendiri selaku peneliti.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama penulis mengerjakan Tugas Akhir maupun selama
menempuh masa studi antara lain:

- Ibu Pujiati selaku ibu penulis yang selalu memberikan dukungan dalam dari berbagai aspek dan menjadi penyemangat untuk segera mendapatkan gelar sarjana ini.
- Bapak Handi Saroso selaku bapak penulis yang memberikan inspirasi serta membiayai segala kebutuhan penulis hingga saat ini.

- Bapak Rully Soelaiman, S.Kom., M.Kom. selaku Dosen Pembimbing yang telah banyak memberikan ilmu, pandangan, nasihat, motivasi, dan bimbingan selama penulis menempuh masa perkuliahan maupun selama pengerjaan Tugas Akhir ini.
- Bapak Dwi Sunaryono, S.Kom., M.Kom. selaku Dosen Pembimbing yang telah memberikan bimbingan dan arahan dalam pengerjaan Tugas Akhir ini.
- Nugrahadi Yanuarso selaku kakak penulis yang juga menjadi penyemangat untuk segera mendapatkan gelar sarjana ini.
- Teman-teman angkatan 2014 yang menemani penulis selama masa perkuliahan.
- Teman-teman dari Pati yang selalu menemani keseharian penulis.
- Seluruh dosen, karyawan, dan teknisi yang sudah memberikan ilmu dan mengisi hari penulis selama masa perkuliahan.

Penulis menyadari masih ada kekurangan pada Tugas Akhir ini sehingga Penulis mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan supaya Tugas Akhir ini menjadi lebih baik. Semoga melalui Tugas Akhir ini penulis dapat memberikan manfaat untuk pembaca.

Surabaya, Juli 2018

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

DAFTAR KODE SUMBER

Halaman ini sengaja dikosongkan

BAB I
PENDAHULUAN

Pada bab ini akan dijelaskan latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi dan sistematika penulisan Tugas Akhir.

1.1 Latar Belakang

Tugas Akhir ini mengacu pada permasalahan klasik *SPOJ Word Play*. Permasalahan yang diangkat adalah permasalahan penyusunan beberapa buah string menjadi sebuah string baru yang diinginkan. Diberikan sejumlah N-K+1 buah subkata yang telah terurut secara leksikografis untuk disusun kembali menjadi sebuah kata utuh yang diinginkan, dengan ketentuan sebagai berikut:

> Input:
> Line pertama berisi integer N, K
> N = panjang kata (huruf)
> K = panjang subkata (huruf)
> Line kedua dan seterusnya berisi N-K+1 buah subkata yang terurut dengan urutan leksikografis.
>
> Output:
> Sebuah kata utuh yang diinginkan.

Untuk menyelesaikan permasalahan di atas, penulis akan menggunakan pendekatan solusi dengan menerapkan algoritma komputasi Graf Hamilton dan penggunaan *Standard Template Library* (STL). Selain dapat menjawab pertanyaan dengan benar, waktu juga menjadi salah satu faktor penting untuk

memberikan gambaran tentang performa dari algoritma yang dirancang.
Hasil dari Tugas Akhir ini diharapkan dapat memberikan gambaran mengenai performa algoritma dengan penerapan algoritma komputasi Graf Hamilton dan penggunaan *Standard Template Library* (STL).

1.2 Rumusan Masalah

Perumusan masalah yang terdapat pada Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana analisa desain algoritma komputasi graf hamilton yang sesuai untuk menyelesaikan permasalahan klasik *SPOJ Word Play*?
2. Bagaimana strategi perancangan dan pengimplementasian algoritma komputasi graf hamilton untuk menyelesaikan permasalahan klasik *SPOJ Word Play*?
3. Bagaimana menguji dan menganalisis performa algoritma komputasi graf hamilton yang telah diimplementasikan untuk menyelesaikan permasalahan *SPOJ Word Play*?

1.3 Batasan Masalah

Batasan masalah yang terdapat pada Tugas Akhir ini adalah sebagai berikut:

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Panjang kata sepanjang N huruf, $3 \leq N \leq 100\,000$.
3. Panjang subkata sepanjang K huruf, $2 \leq K \leq 15$.
4. $K < N$.
5. Banyaknya subkata adalah N-K+1 buah subkata.
6. Semua huruf dituliskan secara kapital.

7. Batas waktu pada program dalam 1 kali percobaan dibawah 0,374 detik.
8. Penyimpanan yang dibutuhkan dalam 1 kali percobaan dibawah 1536 Megabyte

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Menentukan strategi perancangan dan pengimplementasian algoritma komputasi graf hamilton untuk menyelesaikan permasalahan klasik *SPOJ Word Play*.
2. Menganalisa dan mendesain algoritma komputasi graf hamilton yang sesuai untuk menyelesaikan permasalahan klasik *SPOJ Word Play*.
3. Menguji dan menganalisa performa algoritma komputasi graf hamilton yang telah diimplementasikan untuk menyelesaikan permasalahan *SPOJ Word Play*.

1.5 Manfaat

Manfaat dari Tugas Akhir ini adalah sebagai berikut:

1. Mengetahui pemanfaatan penerapan algoritma komputasi Graf Hamilton dan penggunaan *Standard Template Library* (STL) dalam menyelesaikan suatu permasalahan.
2. Melatih kemampuan analisis karakteristik permasalahan yang dapat diselesaikan dengan penerapan algoritma komputasi Graf Hamilton dan penggunaan *Standard Template Library* (STL).

1.6 Metodologi

Metodologi yang digunakan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan Proposal Tugas Akhir
   Tahap pertama dalam proses pengerjaan Tugas Akhir ini adalah menyusun proposal Tugas Akhir. Pada proposal Tugas Akhir ini diajukan sebuah algoritma penyelesaian permasalahan klasik *SPOJ Word Play* dengan penggunaan algoritma dan gagasan solusi yang berkaitan dengan permasalahan tersebut. Luaran dari tahap ini adalah proposal Tugas Akhir.
2. Studi Literatur
   Tahap kedua dalam proses pengerjaan Tugas Akhir ini adalah studi literatur. Pada tahap ini dilakukan pencarian informasi dan studi metode penyelesaian masalah terkait cara-cara meng-optimasi kecepatan compiler C++, penggunaan struktur data yang tepat, dan algoritma yang terkait dengan permasalahan klasik *SPOJ Word Play*. Materi-materi tersebut didapatkan dari buku-buku, paper, internet, serta materi perkuliahan yang terkait. Luaran dari tahap ini adalah daftar pustaka dan referensi.
3. Desain
   Pada tahap ini dilakukan desain rancangan algoritma dan struktur data yang digunakan dalam solusi untuk memecahkan permasalahan klasik *SPOJ Word Play*. Luaran dari tahap ini adalah algoritma dan struktur data yang digunakan dalam implementasi.
4. Implementasi Perangkat Lunak
   Implementasi algoritma adalah tahapan untuk membangun aplikasi yang digunakan dari desain algoritma dan struktur data yang sudah dilakukan pada tahap desain untuk menyelesaikan permasalahan

permasalahan klasik *SPOJ Word Play*. Luaran dari tahap ini adalah implementasi algoritma yang dibangun menggunakan bahasa pemrograman C++ dan menggunakan IDE Dev-C++.

5. Uji Coba Dan Evaluasi
   Tahap pengujian dan evaluasi adalah tahap untuk menguji program yang telah dibuat hasil implementasi algoritma pada permasalahan klasik *SPOJ Word Play*. Pengujian dan evaluasi dilakukan hingga mendapatkan hasil *accepted* dari *Online Judge SPOJ*. Luaran dari tahapan ini adalah status *accepted* dari *Online Judge SPOJ*.
6. Penyusunan Buku Tugas Akhir
   Pada tahap ini dilakukan penyusunan laporan dan buku Tugas Akhir yang berisi dokumentasi hasil pengerjaan Tugas Akhir.

1.7 Sistematika Penulisan

Berikut adalah sistematika penulisan buku Tugas Akhir ini:

1. BAB I: PENDAHULUAN
   Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi dan sistematika penulisan Tugas Akhir.
2. BAB II: DASAR TEORI
   Bab ini berisi dasar teori mengenai permasalahan dan algoritma penyelesaian yang digunakan dalam Tugas Akhir.
3. BAB III: DESAIN
   Bab ini berisi desain algoritma dan struktur data yang digunakan dalam penyelesaian permasalahan.
4. BAB IV: IMPLEMENTASI
   Bab ini berisi implementasi berdasarkan desain algortima yang telah dilakukan pada tahap desain.

5. BAB V: UJI COBA DAN EVALUASI
   Bab ini berisi uji coba dan evaluasi dari hasil implementasi yang telah dilakukan pada tahap implementasi.
6. BAB VI: KESIMPULAN
   Bab ini berisi kesimpulan yang didapat dari hasil uji coba yang telah dilakukan.

BAB II
DASAR TEORI

Pada bab ini akan dijelaskan mengenai dasar teori yang menjadi dasar pengerjaan Tugas Akhir ini

2.1 Deskripsi Umum

Pada subbab ini akan dibahas istilah, definisi, dan deskripsi umum yang digunakan di dalam buku ini. Pembahasan dalam subbab ini memiliki tujuan supaya pembaca dapat lebih mudah memahami dan mengerti isi dari buku ini. Istilah yang digunakan dalam buku ini masuk di dalam Teori Graf dan *Standard Template Library* (STL).

2.2 Graf

Pada subbab 2.2 ini akan dibahas deskripsi Teori Graf yang mencakup Graf Hamilton, teorema-teorema yang ada, hingga pengimplementasiannya pada suatu permasalahan.

2.2.1 Deskripsi Graf

Teori graf atau teori grafik dalam matematika dan ilmu komputer adalah cabang kajian yang mempelajari sifat-sifat "graf" atau "grafik". Ini tidak sama dengan "Grafika". Secara informal, suatu graf adalah himpunan benda-benda yang disebut "simpul" (*vertex* atau *node*) yang terhubung oleh "sisi" (*edge*) atau "busur" (*arc*) [1]. Biasanya graf digambarkan sebagai kumpulan titik-titik (melambangkan "simpul") yang dihubungkan oleh garis-garis (melambangkan "sisi") atau garis berpanah (melambangkan "busur"). Suatu sisi dapat menghubungkan suatu simpul

dengan simpul yang sama. Sisi yang demikian dinamakan "gelang" (*loop*).

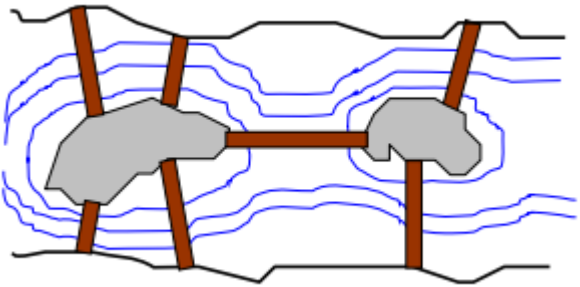
Notasi sebuah graf adalah G = (V, E), dimana:

- V merupakan himpunan tak kosong dari simpul-simpul (*vertices*), misalkan V = {v_1, v_2, ..., v_n}
- E merupakan himpunan sisi-sisi (edges) yang menghubungkan sepasang simpul, misalkan E = {e_1, e_2, ..., e_n}
- Jika graf tersebut mempunyai himpunan sisi yang merupakan himpunan kosong, dinamakan *null graph* atau *empty graph*. [1]

Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf maka graf digolongkan menjadi dua jenis:

1. Graf sederhana (*simple graph*)
   Graf yang tidak mengandung gelang maupun sisi ganda dinamakan graf sederhana.
2. Graf tak-sederhana (*unsimple graph*)
   Graf yang mengandung sisi ganda atau gelang dinamakan graf tak sederhana (*unsimple graph*). Ada dua macam graf tak sederhana, yaitugraf ganda (*multigraph*) dan graf semu (*pseudograph*). Graf ganda adalah graf yang mengandung sisi ganda. Graf semu adalah graf yang mengandung gelang (*loop*).

Sebagai contoh teka-teki yang dikenal dengan nama masalah *Seven Bridges of Konigsberg*. Dalam teka-teki ini terdapat tujuh buah jembatan yang menghubungkan dua pulau dan sebuah sungai, seperti yang ditunjukkan pada Gambar 2.1 berikut. Akan dicari sebuah lintasan yang melewati setiap jembatan tepat satu kali.

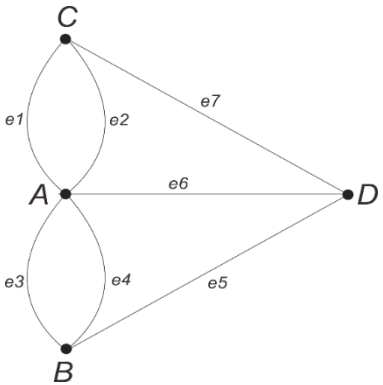


Gambar 2.1. Jembatan Konigsberg [2]

Sebuah metode untuk mencari solusi dari masalah ini adalah dengan membentuk model dari jembatan Konigsberg yang dikenal sebagai multigraph, diperlihatkan pada Gambar 2.2. Sebuah multigraph memiliki dua elemen yaitu himpunan verteks (titik/node) dan himpunan edge (garis) yang menghubungkan antar verteks. Misalkan graf tersebut adalah G (V, E) dengan

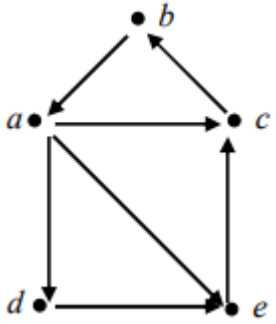
V = {A, B, C, D}

E = {(A,C), (A,C), (A,B), (A,B), (B,D), (A,D), (C,D)}

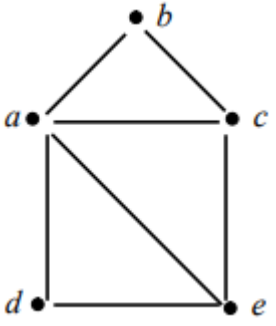


Gambar 2.2. Representasi Graf Jembatan Konigsberg

Multigraph pada jembatan Konigsberg memiliki empat verteks, yang mana keempat verteks tersebut memiliki edge yang meninggalkan verteks tersebut berjumlah ganjil. Maka multigraph jembatan Konigsberg tidak memiliki lintasan Eulerian. Penjelasan lebih lanjut mengenai lintasan Eulerian pada subbab 2.2.2. *Multigraph* yang ditunjukkan pada Gambar 2.4 tidak memiliki panah, sehingga disebut dengan *undirected graph* (graf tak berarah). Sebaliknya, *multigraph* yang memiliki panah disebut dengan *directed graph* (graf berarah) [2], seperti graf pada Gambar 2.3.



Gambar 2.3. Directed Graph



Gambar 2.4. Undirected Graph

1. Graf tak-berarah (*undirected graph*)
   Graf yang sisinya tidak mempunyai orientasi arah disebut tak-berarah. Pada graf tak-berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan. Jadi, (a, c) dan (c, a) adalah sisi yang sama. (Gambar 2.4)

2. Graf berarah (*directed graph* atau *digraph*)
   Graf yang setiap sisinya diberikan orientasi arah disebut sebagai graf berarah. Pada graf berarah, (a, c) dan (c, a) menyatakan dua buah busur yang berbeda, dengan kata lain (a, c) ≠ (c, a). Untuk busur (a, c) simpul a dinamakan simpul asal (*initial vertex*) dan simpul c dinamakan simpul terminal (*terminal vertex*). (Gambar 2.3)

Sedangkan berdasarkan siklusnya, siklus dalam graf akan terbagi menjadi dua yaitu Euler dan Hamilton. Eulerian adalah sebuah siklus dalam graf yang memastikan bahwa dirinya telah melewati semua sisi (*edges*) yang ada dalam graf tersebut. Dan tidak menjadi suatu masalah jika sebuah verteks dilewati sebanyak apapun. Tetapi pada Hamilton adalah sebuah siklus dalam graf yang memastikan bahwa dirinya telah melewati semua verteks dalam graf tersebut dan hanya tepat satu kali, kecuali verteks awal didatangi dua kali. Jika sebuah verteks itu telah dilewati dua atau lebih dalam suatu siklus maka siklus tesebut tidak dapat dikatakan sebagai siklus Hamiltonian.

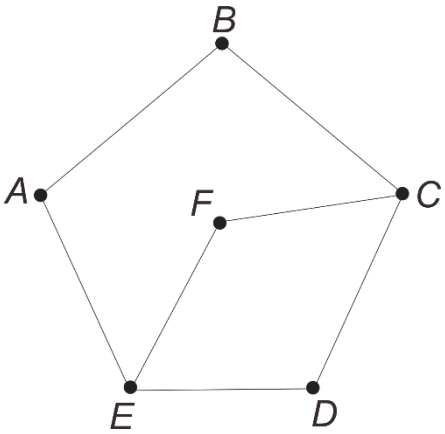
2.2.2 Graf Hamilton

Definisi dari lintasan Hamilton adalah lintasan yang melalui tiap verteks di dalam graf tepat satu kali [3]. Bila lintasan itu kembali ke verteks asal membentuk lintasan tertutup (sirkuit), maka lintasan tertutup itu dinamakan sirkuit Hamilton. Dengan kata lain, sirkuit Hamilton adalah sirkuit yang melalui tiap verteks di dalam graf tepat satu kali, kecuali verteks asal (sekaligus verteks akhir) yang dilalui dua kali. Graf yang memiliki sirkuit Hamilton dinamakan

graf Hamilton, sedangkan graf yang hanya memiliki lintasan Hamilton disebut graf semi-Hamilton.

Untuk menentukan sebuah graf itu adalah Siklus Hamilton atau tidak, pastinya lebih sulit daripada menentukan itu Eulerian. Dan tidak ada cara pasti yang diketahui untuk menentukan itu.

Berikut adalah contoh dari graf Hamilton:

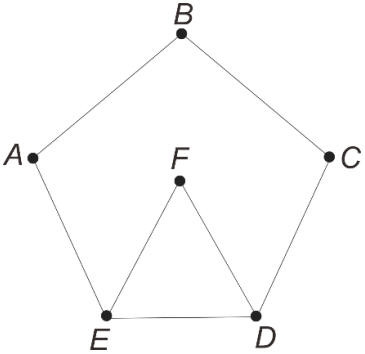


Gambar 2.5. Contoh Lintasan Hamilton

Graf pada Gambar 2.5 adalah contoh graf yang memiliki lintasan Hamilton, tetapi bukan merupakan sirkuit Hamilton. Contoh lintasan Hamilton yang dapat terbentuk adalah:

D – E – F – C – B – A

Graf tersebut tidak memiliki sirkuit Hamilton dikarenakan simpul awal dan simpul akhir tidak dapat bertemu.



Gambar 2.6. Contoh Sirkuit Hamilton

Berbeda dengan graf pada Gambar 2.6, graf tersebut adalah contoh graf yang memiliki lintasan sekaligus sirkuit Hamilton. Sirkuit Hamilton yang dapat terbentuk dari graf tersebut adalah:

A – B – C – D – F – E – A

Graf pada Gambar 2.6 dapat dikatakan memiliki lintasan sekaligus sirkuit Hamilton karena semua simpul dapat dilewati tepat satu kali dan simpul awal dapat bertemu dengan simpul akhir.

2.2.2.1 Teorema Graf Hamilton

Pada subbab ini akan dijelaskan mengenai teorema yang ada pada graf Hamilton. Beberapa sifat-sifat Hamiltonian adalah sebagai berikut:

- Syarat cukup (bukan syarat perlu) supaya graf sederhana G dengan n buah simpul ($n \geq 3$) adalah graf Hamilton ialah jika derajat tiap simpul paling sedikit n/2.
- Setiap graf lengkap adalah graf Hamilton.
- Dalam graf lengkap dengan n buah simpul ($n \geq 3$) terdapat (n-1)/2 buah sirkuit Hamilton.
- Dalam graf lengkap G dengan jumlah simpul $n \geq 3$ dan n ganjil , terdapat (n-1)/2 buah sirkuit Hamilton yang saling lepas (tidak ada sisi yang beririsan). Jika n genap dan $n \geq 4$, maka dalam G terdapat (n-2)/2 buah sirkuit Hamilton.

2.3 Standard Template Library (STL)

Pada subbab ini akan dibahas deskripsi *Standard Template Library* atau yang biasa disingkat STL serta penjelasan mengenai komponen-komponen yang ada pada STL tersebut.

2.3.1 Deskripsi Standard Template Library (STL)

Standard Template Library (STL) adalah pustaka *software* untuk bahasa pemrograman C++ yang mempengaruhi banyak bagian dari Pustaka Standar C++. *Standard Template Library* (STL) ini menyediakan empat komponen yang disebut *algorithm* (algoritma), *containers* (kontainer), *functions* (fungsi), dan *iterators* [4]. Selain menyediakan struktur data dalam bentuk *class*, STL juga memiliki sekumpulan algoritma yang siap dipakai, seperti algoritma

pencarian dan pengurutan. Subbab berikutnya akan menjelaskan mengenai komponen-komponen tersebut.

2.3.2 Algorithm

Sejumlah algoritma untuk menangani aktivitas seperti pencarian dan pengurutan telah disediakan oleh STL, masing-masing digunakan pada tingkat *iterator* tertentu (dan akan bekerja pada semua *container* yang menyediakan *interface* berupa *iterators*). Untuk lebih jelasnya fungsi algoritma ini adalah untuk memproses data di dalam *container*, seperti mencari data atau mengurutkan data.

Beberapa jenis algoritma yang ada pada *Standard Template Library* (STL) antara lain:

- *Non-modifying algorithms*
  Jenis algoritma ini menggunakan *input iterator* dan *forward iterator*.
- *Modifying algorithms*
  Jenis algoritma ini mengubah nilai elemen.
- *Removing algorithms*
  Ini adalah jenis *modifying algorithms* yang menghapus elemen.
- *Mutating algorithms*
  Ini adalah jenis *modifying algorithms* yang mengubah urutan elemen.
- *Sorting algorithms*
  Ini adalah jenis khusus dari *mutating algorithms* yang lebih rumit.
- *Sorted range algorithms*
  Ini adalah jenis algoritma yang beroperasi pada *range* yang telah terurut.

- *Numeric algorithms*
  Algoritma jenis ini menggabungkan elemen numerik.

Walaupun algoritma mem-proses elemen dalam *container*, mereka tidak secara langsung berhadapan dengan *container*, melainkan melalui *iterator*. Dengan desain seperti ini, satu algoritma dapat didefinisikan hanya sekali saja, tetapi bekerja untuk banyak jenis *container*. Beberapa contoh *algorithms* misalnya: *min_element*(), *lower_bound*(), *sort*(), dan *find*().

2.3.3 Container

Berdasarkan urutan datanya, ada dua jenis *containers* yang disediakan oleh STL, yaitu:

- *Sequence Containers*
  Setiap elemen di dalam containers ini memiliki urutan yang pasti, dalam kata lain urutan data sangat penting untuk jenis *containers* ini. Yang masuk dalam kategori ini adalah *vector*, *deque*, dan *list*. [4]
- *Associative Containers*
  Urutan tidak penting dalam *containers* ini. Yang masuk dalam kategori ini adalah *set*, *multiset*, *map*,dan *multimap*. [4]

1. *Container vector* menyimpan elemennya dalam bentuk *array* dinamis. *Vector* memungkinkan akses secara acak untuk setiap elemen di dalamnya (*random access*), tidak harus selalu mulai dari awal (*sequential*). Menulis dan menghapus elemen pada bagian akhir sebuah *vector* berlangsung cepat, jika dibandingkan dengan

menyisipkan atau menghapus elemen di tengah-tengah dan di awal *vector*.

2. *Container deque* (*double-ended queue*) menyimpan elemennya dalam bentuk array dinamis yang dapat bertambah dari elemen paling awal maupun dari elemen paling akhir. Dengan demikian, operasi menulis atau menghapus elemen pada bagian paling awal maupun paling akhir dapat berlangsung dengan cepat.
3. *Container list* menyimpan elemennya dalam bentuk *double linked-list*, dimana setiap elemen-nya memiliki referensi ke elemen sebelum dan elemen sesudahnya. Dengan demikian, setiap elemen di dalam list tidak dapat diakses secara acak, melainkan harus berurut berdasarkan elemen sebelumnya atau sesudahnya. Kelebihan list adalah penambahan dan penghapusan elemen dapat berlangsung dengan cepat di posisi manapun.
4. Pada *container set*, setiap elemen akan diurutkan berdasarkan nilainya, dan tidak boleh ada nilai yang ganda/duplikat.
5. *Container multiset* sama seperti *set*, hanya saja container ini memungkinkan beberapa elemen untuk memiliki nilai yang sama.
6. *Container map* menyimpan elemennya dalam bentuk pasangan *key/value* dengan *key* yang unik [5]. Setiap elemen memiliki *key* yang dijadikan basis pengurutan. Tidak boleh *key* yang ganda/duplikat di *map*.
7. *Container multimap* sama seperti *map*, hanya saja *container* ini memungkinkan beberapa elemen untuk memiliki *key* yang sama.

2.3.4 Iterator

Iterator adalah *object* yang dipakai untuk menjelajahi elemen dalam *container*. *Iterator* menyediakan operator seperti:

- *Operator* *
  Untuk mendapatkan elemen yang sedang berada di posisi saat ini.
- *Operator* ++
  Untuk menjelajahi ke elemen berikutnya.
- *Operator* == dan !=
  Untuk perbandingan apakah dua iterator sedang merujuk ke posisi yang sama.
- *Operator* =
  Untuk mengubah posisi yang sedang dirujuk iterator.

C++ memungkinkan kita untuk mengubah makna *operator* bagi sebuah *object*, dan STL mengubah *operator* yang biasanya dipakai untuk *pointer*.

Setiap *container* menyediakan dua jenis *iterator*, yaitu:

- *container::iterator*
  Untuk melakukan navigasi dalam modus baca/tulis.
- *container::const_iterator*
  Untuk melakukan navigasi dalam modus hanya baca saja.

2.4 Permasalahan Word Play pada SPOJ

Permasalahan yang diangkat dalam Tugas Akhir ini bersumber dari *online judge SPOJ* yaitu permasalahan *Word Play* [6]. Di dalam permasalahan ini diketahui bahwa ada sebuah kata tersusun dari N huruf, yang terpisah menjadi N-K+1 buah sub kata. Dengan K adalah panjang subkata (tersusun dari K huruf). Subkata-subkata ini terurut secara leksikografis. Jawaban yang diinginkan adalah sebuah kata utuh yang tersusun dari subkata-subkata yang telah diberikan. Uji kasus akan dibuat secara acak.

Format masukan untuk permasalahan *Word Play* dapat dilihat sebagai berikut:

1. Baris pertama adalah *integer* N dan *integer* K. N untuk merepresentasikan panjang kata. Sedangkan K untuk merepresentasikan panjang subkata.
2. Baris kedua dan seterusnya berisi N-K+1 buah subkata yang sudah tersusun secara leksikografis dan semua ditulis dalam huruf kapital.

Format keluaran berisi sebuah kata yang tersusun dari N-K+1 buah subkata yang telah diberikan. Contoh dari format masukan dan keluaran dapat dilihat pada Gambar 2.7 berikut. Batasan-batasan pada permasalahan *Word Play* adalah sebagai berikut:

1. $3 \leq N \leq 100000$
2. $2 \leq K \leq 15$
3. $K \leq N$

Input:

```
6 3
ANA
ANA
BAN
NAN
```

Output:

```
BANANA
```

Gambar 2.7. Contoh Input-Output Permasalahan Word Play

2.5 Penyelesaian Permasalahan Word Play

Pada subbab ini akan dijelaskan bagaimana menyelesaikan permasalahan *Word Play* dengan penjelasan permasalahan seperti pada subbab 2.4.

Teori graf Hamilton akan diimplementasikan pada permasalahan ini, lebih tepatnya adalah graf berarah yang mengandung lintasan Hamilton. Pada deskripsi soal diketahui bahwa terdapat input sejumlah N-K+1 buah subkata berupa string, subkata-subkata tersebut diumpamakan sebagai simpul atau verteks dalam sebuah graf. Pada graf Hamilton, lintasan Hamilton dapat terbentuk apabila semua simpul dapat dilewati dengan masing-masing simpulnya dilewati tepat hanya satu kali. Apabila subkata-subkata yang menjadi input diumpamakan sebagai simpul pada graf Hamilton, maka semua subkata harus digunakan dan masing-masing subkata hanya boleh digunakan tepat satu kali.

Antara satu simpul (subkata) dengan lainnya dapat dikatakan terhubung apabila K-1 huruf belakang simpul (subkata) pertama sama dengan K-1 huruf depan simpul (subkata) kedua.Sebagai contoh ANA dan NAN dapat dikatakan terhubung, karena 2 huruf belakang subkata pertama (ANA) sama dengan 2 huruf depan subkata kedua (NAN). Apabila ANA dan NAN digabungkan akan membentuk kata ANAN.

Dari contoh masukan pada Gambar 2.7, lintasan Hamilton yang diharapkan terjadi adalah:

BAN – ANA – NAN – ANA

Pembahasan lebih lanjut mengenai terbentuknya lintasan Hamilton tersebut ada pada bab 3.

Untuk menampung *input* sejumlah N-K+1 subkata yang berupa *string*, digunakan *container map(key,value)*. *String* subkata menjadi *Key*, dan *value* diisi oleh *integer* jumlah subkata yang sama.

Digunakan pengulangan dan juga algoritma pencarian (*lower_bound* dan *upper_bound*), dengan kompleksitas waktu mendekati $\log_2 n$, untuk mencari simpul yang dapat digabungkan. Sehingga output yang terjadi dari kemungkinan lintasan Hamilton tersebut adalah kata BANANA.

Halaman ini sengaja dikosongkan

BAB III
DESAIN

Pada bab ini akan dijelaskan desain algoritma yang digunakan dalam pengerjaan permasalahan pembuktian algoritma dan pengerjaan Tugas Akhir ini.

3.1 Desain Penyelesaian Permasalahan Word Play

Pada subbab ini akan dijelaskan desain penyelesaian permasalahan Word Play.

3.1.1 Definisi Umum Sistem

Sistem akan menerima masukan berupa dua buah bilangan N dan K, yang masing-masing mewakili N untuk menyimpan *integer* panjang kata yang diinginkan, dan K untuk menyimpan *integer* panjang subkata yang akan menjadi masukan nantinya. Selanjutnya sistem akan menerima masukan berupa *string* sebanyak N-K+1 dan sepanjang K. Semua masukan sesuai dengan format yang sudah ditentukan pada subbab 2.4 sebelumnya, permasalahan ini kemudian diimplementasikan menjadi graf berarah yang mengandung lintasan Hamilton. Masukan-masukan berupa *string* diumpamakan sebagai simpul (verteks) dalam graf seperti yang telah dijelaskan pada subbab 2.5.

Untuk menyimpan masukan *string* tersebut, digunakan *container map* seperti yang telah dijelaskan pada subbab 2.3.3. masukan *string* disimpan sebagai *Key* dan *Value* diisi nilai *integer* jumlah *string* yang sama. Sebagai contoh masukan seperti pada Gambar 2.7 pada subbab 2.4,

penyimpanan masukan pada *container map* adalah seperti berikut:

Tabel 3.1. Tabel Penyimpanan Data Pada Container Map

Key	Value
ANA	2
BAN	1
NAN	1

Digunakan *container map* karena sebisa mungkin *input* pada permasalahan ini harus berbeda dengan tujuan untuk menekan waktu pencarian, tetapi juga tanpa menghapuskan nilai input yang sama. Jadi, dikarenakan sifat *container map* yang dapat menerima masukan ganda (sama) dengan syarat *key* yang disimpan harus unik. Maka masukan *string* ANA yang kedua tetap dianggap, tetapi harus dengan menambahkan *value* dari *key* ANA yang telah tersimpan sebelumnya, seperti terlihat pada tabel data masukan pada Tabel 3.1.

Data masukan tadi kemudian diproses, awal mula yang akan diproses adalah elemen pertama (ANA), *iterator full* diarahkan ke elemen tersebut. Diambil sejumlah K-1 huruf depan dan K-1 huruf belakang, disimpan untuk dibandingkan nantinya dengan *key* dari elemen lain. Setiap elemen yang telah digunakan akan langsung dihapus. Kemudian dicari elemen yang dapat terhubung dengan ketentuan seperti yang telah dijelaskan pada subbab 2.5 sebelumnya. Ada dua kemungkinan dalam pencarian elemen selanjutnya:

1. Apabila ditemukan elemen selanjutnya yang dapat terhubung sesuai dengan ketentuan tersebut, *iterator full* akan berubah mengarah ke elemen selanjutnya.

2. Apabila tidak ditemukan elemen selanjutnya, maka *iterator full* akan diarahkan ke elemen pertama lagi.

Masukan	Baris pertama berisi *integer* N dan *integer* K Baris kedua dan seterusnya berisi *string* N-K+1 buah subkata yang dimasukkan dalam *container map*.
Keluaran	Sebuah kata yang sepanjang N huruf.

```
main()
1   input N
2   input K
3   for i = 0 to N-K+1
4     input stock []
5   while !stock.empty ()
6     full = stock.begin ()
7     temp = full
8     for i = 0 to K-1
9       lastused [i] = temp [i]
10      current [i] = temp [i+1]
11    result += temp [K-1]
12    while true
13      delete stock [temp]
14      left = stock.lower_bound (current)
15      right = stock.upper_bound (current)
16      if left == right
17        word = result + word
18        break
19      else
20        full = left
21        for i = 1 to K-1
22          current [i-1] = left (i)
23        result += left (K-1)
24  word = lastused + word
25  output word
```

Gambar 3.1. Pseudocode Sistem

Setelah *iterator full* mendapatkan elemen selanjutnya, kemudian ulang lagi proses di atas, seperti mengambil K-1 huruf depan, mengambil K-1 huruf belakang, dan seterusnya. Proses pengulangan akan terus berulang hingga *container map* kosong. Format keluaran sesuai dengan subbab 2.4. *Pseudocode* ditunjukkan oleh Gambar 3.1 diatas.

3.1.2 Desain Algoritma

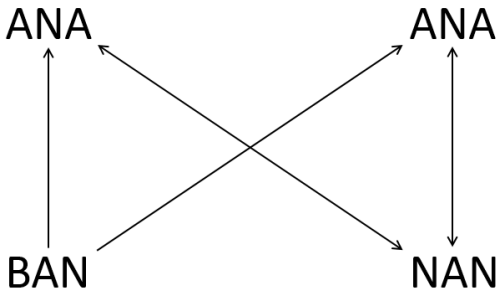
Sistem terdiri dari satu fungsi utama *main*(). Proses di dalam fungsi *main*() dibagi menjadi beberapa proses, antara lain proses *input*, proses penentuan *head* graf, proses penghapusan elemen, proses pencarian *node* dan proses mencetak *output*. Proses-proses di dalam fungsi *main*() tersebut menggunakan *Standard Template Library* (STL). Seperti pada subbab 2.3, *Standard Template Library* (STL) ini menyediakan empat buah komponen yang disebut *algorithm* (algoritma), *containers* (kontainer), *functions* (fungsi), dan *iterators*. Teori graf juga akan diimplementasikan pada permasalahan *Word Play*.

3.1.2.1 Desain Graf pada Soal Word Play

Seperti yang sudah dijelaskan pada subbab 2.5, teori graf akan diimplementasikan pada permasalahan *Word Play*, lebih spesifik adalah graf berarah dengan lintasan Hamilton. Tiap masukan subkata diumpakan sebagai simpul/verteks yang ada dalam graf, sedangkan sisi (*edges*) graf terbentuk dari kemungkinan masing-masing subkata untuk terhubung dengan subkata lain. Subkata satu dengan subkata lain dapat dikatakan terhubung apabila K-1 huruf belakang subkata pertama sama

dengan K-1 huruf depan subkata kedua, sebagai contoh subkata A<u>NA</u> dan <u>NA</u>N, apabila digabungkan akan menjadi A<u>NA</u>N.

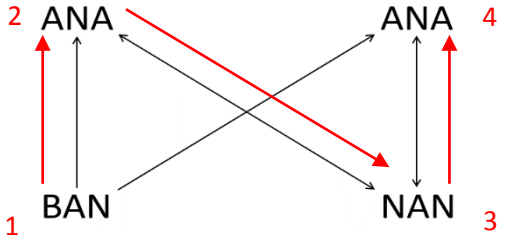
Dengan menggunakan contoh masukan yang telah dijelaskan pada subbab 2.4 dan pada Gambar 2.7, untuk pengimplementasian graf Hamilton pada permasalahan *Word Play* lebih jelasnya perhatikan Gambar berikut ini.



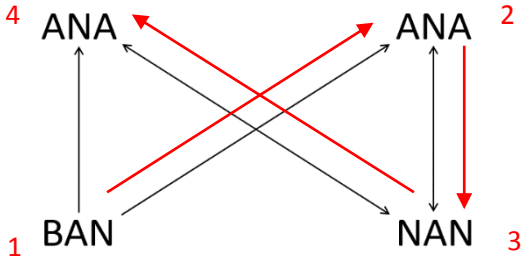
Gambar 3.2. Penggambaran Input Sebagai Node Graf

Dapat dilihat pada Gambar 3.2, setiap masukan subkata diumpamakan sebagai simpul pada graf. Sisi yang digambarkan dengan garis berpanah menunjukkan bahwa graf tersebut merupakan graf berarah. Arah panah menunjukkan bahwa simpul asal dan simpul tujuan dapat dihubungkan. Apabila yang terbentuk panah satu arah, itu berarti bahwa simpul asal dan simpul tujuan dapat dihubungkan, tetapi tidak berlaku sebaliknya. Apabila terbentuk panah dua arah, berarti bahwa posisi simpul asal dan simpul tujuan dapat dibalik dan tetap dapat dihubungkan.

Dari Gambar 3.2, kemungkinan-kemungkinan yang dapat terjadi agar terbentuk lintasan Hamilton adalah seperti pada Gambar 3.3 dan Gambar 3.4 berikut.



Gambar 3.3. Kemungkinan Lintasan Hamilton Pertama



Gambar 3.4. Kemungkinan Lintasan Hamilton Kedua

Kedua kemungkinan tersebut apabila dituliskan akan sama-sama menghasilkan urutan: BAN – ANA – NAN – ANA dan apabila dihubungkan akan menghasilkan kata BANANA.

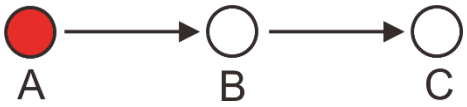
3.1.2.2 Proses Input

Proses *input* ini menggunakan *containers map* yang ada pada *Standard Template Library* (STL) untuk menampung masukan berupa *string*. Parameter yang digunakan pada *map* adalah *map<key,value>*, dimana

parameter pertama adalah *string* dan akan digunakan sebagai *key*, sedangkan parameter kedua adalah *integer* dan akan digunakan sebagai *value* dari setiap *key*-nya (setiap elemen *map*). Urutan elemen yang ada di dalam *container map* tergantung oleh urutan *input* yang diberikan. Ketika menginputkan, apabila belum ada *key* pada *container map* yang sama seperti *string* yang diinputkan, maka akan dibuat elemen baru dengan *string input* sebagai *key*-nya, dan *value* secara default akan diisi *integer* 1. Apabila sudah ada *key* pada *container map* yang sama seperti *string* yang diinputkan, maka tidak akan terjadi penambahan *key* baru, hanya penambahan pada *value* dari *key* yang sama seperti *string* yang diinputkan (*value* += 1). Struktur penyimpanan *container map* lebih jelasnya seperti yang dijelaskan pada subbab 3.1.1.

3.1.2.3 Proses Penentuan Head Graf

Head graf yang dimaksud disini adalah simpul atau *node* awal pada graf yang akan dibentuk. Seperti yang telah dijelaskan pada subbab 3.1.2.1, setiap masukan *string* akan diumpamakan sebagai simpul pada graf. Lebih jelasnya perhatikan graf pada Gambar 3.5 berikut.



Gambar 3.5. Head Graph

Simpul atau *node* yang berwarna merah (A) adalah yang dimaksud dengan *head* graf. *Head graf* akan diambil dari *key* elemen paling awal yang ada di dalam *container map*.

3.1.2.4 Proses Penghapusan Elemen

Setiap elemen yang telah digunakan atau diproses, akan langsung dihapus. Ini dilakukan agar tidak terjadi kesalahan pemilihan antara elemen yang telah digunakan dan belum, tanpa memerlukan pemberian *flag* pada tiap elemennya. Apabila *value* elemen berisi nilai *integer* lebih dari 1 (*value* > 1), maka penghapusan elemen hanya akan dilakukan dengan mengurangi *value*-nya saja (*value* = *value* - 1). Tapi apabila *value* elemen berisi nilai *integer* 1 (*value* == 1), maka akan dilakukan penghapusan elemen (penghapusan *key* dan *value*).

3.1.2.5 Proses Pencarian Node

Setelah *head* graf didapatkan, kemudian dicari *node* lanjutan yang dapat terhubung dengan *head* graf. Antara satu *node* dan lainnya dapat dikatakan terhubung jika memenuhi ketentuan seperti yang telah dijelaskan pada subbab 3.1.2.1, bahwa K-1 huruf belakang *node* pertama harus sama dengan K-1 huruf depan *node* selanjutnya. Proses pencarian *node* ini menggunakan prinsip *binary search*, didukung dengan *sorted range algorithm* yang ada pada *Standard Template Library* (STL), *lower_bound*() dan *upper_bound*().

Proses *binary search* yang dilakukan oleh fungsi *lower_bound*() dan *upper_bound*() mula-mula diawali dengan menentukan batas bawah dan batas atas. Karena semesta huruf yang digunakan adalah huruf kapital A hingga Z, maka huruf A diambil sebagai batas bawah dan dan Z sebagai batas atas. Parameter yang digunakan kedua fungsi tersebut adalah *key*, dimana parameter *key* diisi oleh K-1 huruf belakang *node* awal ditambahkan

satu huruf sebagai batas bawah dan batas atas. Jadi, implementasi kedua fungsi tersebut adalah *lower_bound*() menjadi *lower_bound*([K-1 huruf belakang *node* awal]+A) dan fungsi *upper_bound*() menjadi *upper_bound*([K-1 huruf belakang *node* awal]+Z).

Setelah parameter awal ditentukan, kemudian dilakukan proses pencarian *node* lanjutan dengan cara mencari *node* yang berada pada urutan ke n/2, dimana n adalah jumlah *node* dalam *container* (n disini berarti N-K+1). Apabila node n/2 tersebut lebih kecil (secara leksikografis) dari parameter *key*, maka *node* n/2 menjadi batas bawah pencarian yang baru. Apabila lebih besar, maka *node* n/2 menjadi batas atas pencarian yang baru. Proses tersebut dilakukan sebanyak $\log_2$ (n). Akan tetapi, apabila node n/2 sama dengan parameter *key* yang dicari, maka tidak perlu dilakukan pencarian *node* n/2 lagi, karena *node* yang dicari telah ditemukan.

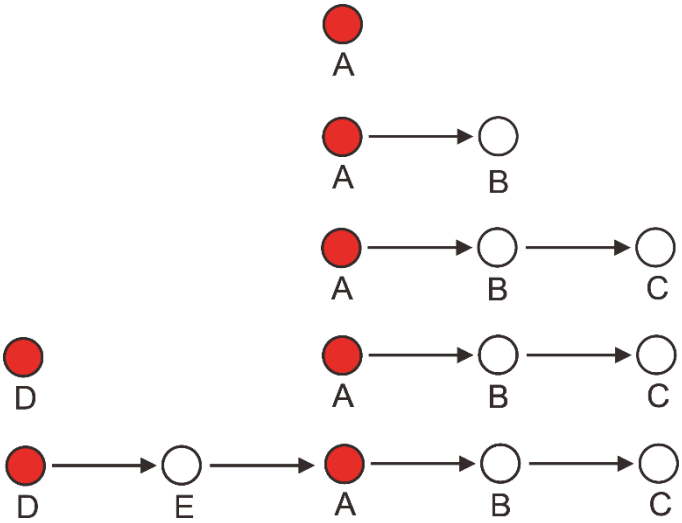
Jalankan dan ulangi proses pencarian *node* tersebut hingga tidak ada lagi *node* yang tersisa atau ada yang masih tersisa tetapi tidak ada yang dapat dihubungkan dengan *node* sebelumnya. Apabila pencarian *node* dapat dilakukan langsung hingga semua *node* dalam *container map* digunakan, maka proses pencarian dianggap selesai dan *output* sudah bisa didapatkan. Tapi apabila kemungkinan kedua yang terjadi, dimana masih ada *node* yang tersisa tetapi tidak ada yang dapat dihubungkan dengan *node* sebelumnya, maka ulangi lagi langkah penentuan *head* graf dengan sisa elemen setelah terjadi proses penghapusan, proses penghapusan, dan juga

proses pencarian *node* kembali. Begitu seterusnya hingga semua elemen dalam *container* terpakai.

Sebagai contoh terdapat 5 buah *node*: A, B, C, D, dan E. Dengan ketentuan A dapat terhubung dengan B, B dapat terhubung dengan C, D dapat terhubung dengan E, dan E dapat terhubung dengan A. Kemungkinan lintasan hamilton yang bisa terbentuk adalah:

D – E – A – B – C

Gambar 3.6 menunjukkan ilustrasi proses pencarian *node*-nya.



Gambar 3.6. Ilustrasi Proses Pencarian Node

Untuk kompleksitas waktu proses ini, karena proses proses perulangan dilakukan sebanyak jumlah masukan subkata, maka kompleksitas waktu menjadi N-K+1. Sedangkan proses *binary search* yang dilakukan oleh fungsi *lower_bound* dan *upper_bound* memiliki kompleksitas waktu $\log_2$ n, dimana n mewakili N-K+1. Jadi kompleksitas waktu program adalah $O((N-K+1)\log_2(N-K+1))$.

3.1.2.6 Proses Mencetak Output

Proses mencetak *output* ini akan dijalankan apabila proses pencarian *node* selesai dilakukan dan semua elemen di dalam *container map* sudah digunakan. *Output* akan dicetak dengan fungsi standard *cout*, dan merupakan proses terakhir yang akan dijalankan.

Halaman ini sengaja dikosongkan

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas tentang implementasi yang dilakukan berdasarkan apa yang sudah dirancang pada bab sebelumnya.

4.1 Lingkungan Implementasi

Lingkungan implementasi dalam pembuatan Tugas Akhir ini meliputi perangkat keras dan perangkat lunak yang digunakan untuk melakukan penyelesaian dari permasalahan yang sudah ada adalah sebagai berikut:

1. Perangkat Keras:
    - *Processor*: Intel(R) Core(TM) i5-4200U CPU @ 1.60 GHz 2.30 GHz.
    - *Random Access Memory*: 4,00 GB.
2. Perangkat Lunak:
    - *Operating System*: Windows 8.1 Pro 64 bit.
    - IDE: Orwell Bloodshed Dev-C++ 5.11.
    - *Compiler*: g++ (TDM-GCC 4.9.2 64-bit).
    - Bahasa Pemrograman: C++.

4.2 Implementasi Penyelesaian Permasalahan Word Play

Pada subbab ini akan dijelaskan mengenai implementasi dari penyelesaian permasalahan *Word Play* yang telah dibuat pada bab 3.

4.2.1 Penggunaan Library, Variabel, dan Template

Pada subbab ini akan dibahas penggunaan *library*, variabel dan *template* yang digunakan dalam sistem.

4.2.1.1 Library

Potongan kode yang menyatakan penggunaan *library* dapat dilihat pada Kode Sumber 4.1.

```
1  #include <bits/stdc++.h>
2  using namespace std;
```

Kode Sumber 4.1. Kode Penggunaan Library

Untuk menyelesaikan permasalahan *WORD-Word Play* dibutuhkan *library* bits/stdc++.h dan juga *namespace std*. *Library* bits/stdc++.h bukan merupakan *file header standard* dari GNU C++, tetapi mencakup banyak *library standard*. Dan *namespace std* mencakup fungsi/*class*/*file* yang umum digunakan seperti perintah *cin* dan *cout*.

4.2.1.2 Variabel

Pada Tabel 4.1.a, Tabel 4.1.b dan Tabel 4.1.c berikut dijelaskan mengenai variabel-variabel yang akan digunakan dalam implementasi program.

Tabel 4.1.a. Tabel Penggunaan Variabel

No	Nama Variabel	Tipe	Penjelasan
1	n	int	Digunakan untuk menyimpan nilai N, dimana N adalah panjang kata (jumlah huruf) yang diinginkan.

Tabel 4.1.b. Tabel Penggunaan Variabel

No	Nama Variabel	Tipe	Penjelasan
2	k	int	Digunakan untuk menyimpan nilai K, dimana K adalah panjang subkata (jumlah huruf) yang digunakan sebagai masukan.
3	lastused	string	Digunakan untuk menyimpan subkata terakhir yang digunakan atau diumpamakan sebagai simpul awal graf.
4	word	string	Digunakan untuk menyimpan string yang menjadi keluaran program.
5	current	string	Digunakan untuk menyimpan K-1 huruf belakang subkata yang sedang ditunjuk oleh iterator (subkata yang sedang digunakan dalam program).
6	temp	string	Digunakan untuk menyimpan subkata yang sedang ditunjuk oleh iterator.

Tabel 2.1.c. Tabel Penggunaan Variabel

No	Nama Variabel	Tipe	Penjelasan
7	result	string	Digunakan untuk menyimpan string yang menjadi keluaran sementara, sebelum dimasukkan pada variabel *word*.
8	left	map iterator	Digunakan untuk menunjuk pada suatu elemen dalam *container map* yang posisinya berada di sebelah kiri (sebelum) elemen yang sedang aktif dan telah dibandingkan dengan variabel *current*.
9	right	map iterator	Digunakan untuk menunjuk pada suatu elemen dalam *container map* yang posisinya berada di sebelah kanan (setelah) elemen yang sedang aktif dan telah dibandingkan dengan variabel *current*.
10	full	map iterator	Digunakan untuk menunjuk pada suatu elemen dalam *container map* yang kemudian *Key*-nya akan dimasukkan dalam variabel *temp*.

Implementasi variabel yang dijelaskan oleh Tabel 4.1, ditunjukkan oleh Kode Sumber 4.2 berikut.

```
1  int n, k;
2  string lastused, word, current, temp;
3  map <string, int> :: iterator left, right, full;
4  string result;
```

Kode Sumber 4.2. Implementasi Variabel

4.2.1.3 Template

Untuk *template* yang digunakan adalah *Standard Template Library* (STL) yang mencakup empat komponen yaitu *algorithm* (algoritma), *containers* (kontainer), *functions* (fungsi), dan *iterators*. *Algorithms* (algoritma) STL yang digunakan adalah sebagai berikut:

- *lower_bound (key)*
  iterator untuk mencari elemen paling pertama yang mengandung *key* seperti *key* yang diberikan.
- *upper_bound (key)*
  iterator untuk mencari elemen paling terakhir yang mengandung *key* seperti *key* yang diberikan.

Untuk *containers*, *containers* yang digunakan adalah *containers map*. *Containers map* ini menyimpan elemennya dalam bentuk pasangan *key/value*. Setiap elemen memiliki *key* yang menjadi basis pengurutan.

Penggunaan iterators untuk penyelesaian permasalahan WordPlay telah dijelaskan pada subbab 4.2.1.2. *Iterator*

ini berguna untuk menjelajahi elemen-elemen yang ada pada *containers map*.

4.2.2 Implementasi Proses Input

Proses *input* digunakan untuk mengambil dan menampung masukan pada *container map*. Sesuai dengan *Pseudocode* pada subbab 3.1.1, proses *input* ini akan diimplementasikan seperti Kode Sumber 4.3. Kegunaan dari masing-masing variabel telah dijelaskan pada subbab 4.2.1.2. Pada Kode Sumber 4.3 berikut juga terdapat deklarasi *container map* yang sebagai penampung *string* masukan.

```
1   int n, k;
2   cin >> n >> k;
3
4   map <string, int> stock;
5   string lastused, word, current, temp;
6
7   lastused.resize (k-1);
8   current.resize (k-1);
9
10  for (int i = 0; i < n-k+1; i++) {
11      cin >> temp;
12      stock [temp]++;
13  }
```

Kode Sumber 4.3. Implementasi Proses Input

4.2.3 Implementasi Proses Penentuan Head Graf

Seperti yang telah dijelaskan pada subbab 3.1.2.3, proses penentuan *head* graf dilakukan dengan mengambil *key* elemen pertama yang ada pada *container map*. Variabel yang bertugas menyimpan elemen pertama tersebut adalah variabel sekaligus *iterator full*. Kode Sumber 4.4 menunjukkan implementasi prosesnya.

```
1   map <string, int> :: iterator left, right, full;
2   string result;
3
4   full = stock.begin ();
5   temp = full -> first;
6
7   for (int i = 0; i < k-1; i++) {
8       lastused [i] = temp [i];
9       current [i] = temp [i+1];
10  }
```

Kode Sumber 4.4. Implementasi Proses Penentuan Head Graf

4.2.4 Implementasi Proses Penghapusan Elemen

Penghapusan elemen dilakukan pada elemen yang telah selesai digunakan, agar tidak terjadi keambiguan antara elemen yang telah digunakan dan elemen yang belum digunakan, tanpa perlu memberikan flag pada tiap elemennya. Ketentuan penghapusan seperti yang telah dijelaskan pada subbab 3.1.2.4. Implementasi proses penghapusan seperti pada Kode Sumber 4.5 berikut.

```
1   stock [temp]--;
2
3   if (!stock [temp]) {
4       stock.erase (full);
5   }
```

Kode Sumber 4.5. Implementasi Proses Penghapusan Elemen

4.2.5 Implementasi Proses Pencarian Node

Pencarian *node* ini dilakukan melalui dua kali proses *looping*/perulangan dengan menggunakan *while*. Penjelasan proses pencarian *node* lebih jelasnya seperti pada subbab 3.1.2.5. Implementasinya seperti pada Kode Sumber 4.6 berikut.

```
1   while (!stock.empty ()) {
2       map <string, int> :: iterator left, right, full;
3       string result;
4       full = stock.begin ();
5       temp = full -> first;
6       for (int i = 0; i < k-1; i++) {
7           lastused [i] = temp [i];
8           current [i] = temp [i+1];
9       }
10      result += temp [k-1];
11      while (1) {
12          temp = full -> first;
13          stock [temp]--;
14          if (!stock [temp]) {
15              stock.erase (full);
16          }
17          left = stock.lower_bound (current + "A");
18          right = stock.upper_bound (current + "Z");
19          if (left == right) {
20              word = result + word;
21              break;
22          }
23          else {
24              full = left;
25              for (int i = 1; i < k; i++){
26                  current [i-1] = (left -> first).at (i);
27              }
28              result += (left) -> first.at (k-1);
29          }
30      }
31  }
```

Kode Sumber 4.6. Implementasi Proses Pencarian Node

4.2.6 Implementasi Proses Mencetak Output

Proses pencetakan *output* akan dilakukan apabila proses pencarian *node* sudah dilakukan dan semua elemen pada *container map* telah digunakan. Variabel yang berfungsi untuk menyimpan hasil *output* adalah variabel *word*. Kode Sumber 4.7 menunjukkan implementasi proses mencetak output.

```
1    word = lastused + word;
2    cout << word << endl;
```

Kode Sumber 4.7. Implementasi Proses Mencetak Output

BAB V
UJI COBA DAN EVALUASI

Pada bab ini dijelaskan tentang uji coba dan evaluasi hasil implementasi yang dilakukan pada Tugas Akhir ini.

5.1 Lingkungan Uji Coba

Pada subbab ini akan dijelaskan tentang lingkungan uji coba yang dilakukan pada Tugas Akhir ini. Bahasa yang digunakan pada uji coba ini adalah bahasa pemrograman C++. *Integrated development Environment* atau IDE yang digunakan adalah Orwell Bloodshed Dev-C++ 5.11 dengan *compiler* g++ (TDM-GCC 4.9.2 64-bit). Sistem Operasi yang digunakan adalah Windows 8.1 Pro 64 bit. Perangkat keras yang digunakan adalah prosesor Intel(R) Core(TM) i5-4200U CPU @ 1.60 GHz 2.30 GHz. *Random Access Memory* atau RAM yang digunakan berukuran 4.00 GB.

5.2 Uji Coba Kebenaran

Pada subbab ini akan dijelaskan uji coba yang dilakukan untuk menguji desain dan implementasi yang sudah dibuat untuk menyelesaikan Tugas Akhir ini.

5.2.1 Uji Coba Kebenaran Penyelesaian Permasalahan Word Play

Uji coba kebenaran dilakukan dengan cara mengirimkan kode sumber program ke situs penilaian *SPOJ*. Hasil uji coba dengan waktu terbaik yang didapatkan dari situs penilaian *SPOJ* dapat dilihat pada Gambar 5.1.

Gambar 5.1. Hasil Accepted SPOJ

Gambar 5.2. Ranking SPOJ

Dari Gambar 5.1, dapat dilihat bahwa kode sumber yang telah dikirim mendapatkan hasil keluaran *Accepted*. Dan Gambar 5.2 merupakan *ranking accepted SPOJ*. Waktu yang dibutuhkan sistem untuk menyelesaikan soal *Word Play* adalah 0.62 detik dan memori yang dibutuhkan adalah 23 MB. Selanjutnya akan dijelaskan langkah-langkah sistem dalam melakukan penyelesaian soal dan hasil dari uji coba sederhana akan dibandingkan dengan keluaran sistem. Uji coba kasus sederhana akan diselesaikan mengacu dengan langkah-langkah yang sudah dijelaskan pada subbab 2.5 dan mengacu dengan ketentuan-ketentuan *input* yang sudah dijelaskan pada subbab 2.4. Uji coba kasus sederhana yang akan dilakukan pada program adalah sebagai berikut.

5.2.1.1. Uji Coba Kasus

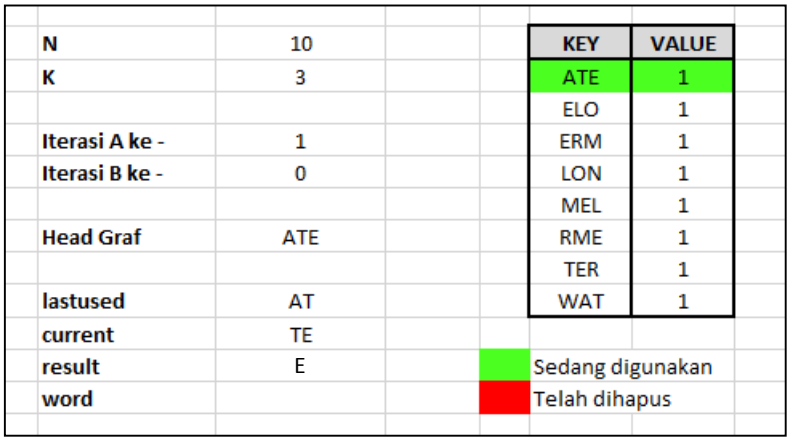
Kasus sederhana yang digunakan sebagai uji coba kasus adalah sebuah kasus dimana Ivana memiliki sebuah kata yang tersusun dari 10 buah huruf (N = 10). Ivana memecah kata tersebut menjadi subkata yang tersusun dari masing-masing 3 huruf (K = 3). Subkata yang disusun berjumlah N–K+1 buah, dalam kasus ini berarti berjumlah 10 – 3 + 1 = 8 buah.

Format uji coba masukan dari masalah yang akan diselesaikan dapat dilihat pada gambar 5.3. Langkah selanjutnya setelah memasukkan kasus uji coba sederhana seperti pada Gambar 5.3 adalah membuat urutan perubahan data yang terjadi selama proses pengerjaan. Yang proses utama adalah proses pencarian *node* yang dilakukan dengan menggunakan dua kali *looping* dengan *while*, *looping* pertama (lebih lanjut disebut *loop* A) dan *looping* kedua (lebih lanjut disebut *loop* B), *loop* B berada di dalam *loop* A. Sebagai permulaan sebelum input diproses lebih lanjut,

```
1  10 3
2  ATE
3  ELO
4  ERM
5  LON
6  MEL
7  RME
8  TER
9  WAT
```

Gambar 5.3. Input Uji Coba Kasus Pertama

diperlukan proses penentuan *head* graf. Mengingat *head* graf ditentukan berdasarkan *key* elemen pertama pada *container map*, jadi yang menjadi *head* graf pada uji kasus ini adalah ATE. Perubahan data variabel pada tiap iterasi dan perubahan data elemen pada *container map* akan divisualisasikan sebagai berikut.



N	10
K	3
Iterasi A ke -	1
Iterasi B ke -	0
Head Graf	ATE
lastused	AT
current	TE
result	E
word	

KEY	VALUE
ATE	1
ELO	1
ERM	1
LON	1
MEL	1
RME	1
TER	1
WAT	1

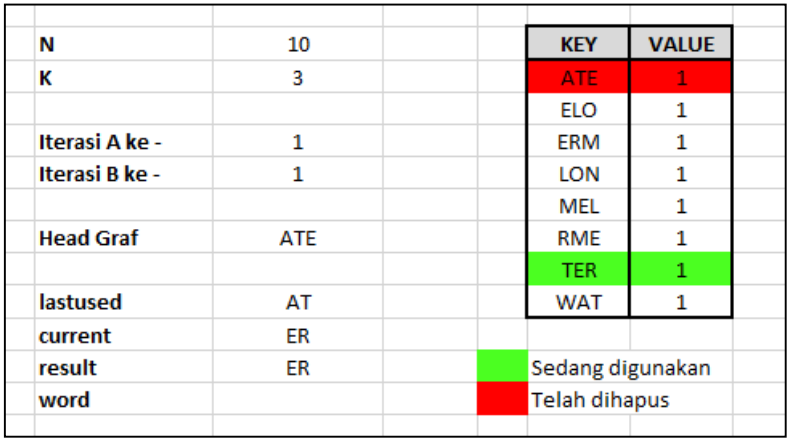
Sedang digunakan

Telah dihapus

Gambar 5.4. Perubahan Variabel pada Iterasi A ke-1 dan B ke-0 Uji Kasus 1

Iterasi Gambar 5.4 adalah iterasi pertama pada *loop* A dan belum memasuki *loop* B. *Key* elemen ATE telah ditentukan sebagai *head* graf. Variabel *lastused* digunakan untuk menyimpan K-1 huruf depan dari *head* graf, variabel *current* digunakan untuk menyimpan K-1 huruf belakang dari *key* elemen yang saat ini sedang digunakan, untuk iterasi sekarang yang sedang digunakan adalah *head* graf ATE. Variabel *result* digunakan untuk menyimpan *output* sementara sebelum dimasukkan ke variabel *word*, variabel *result* diisi oleh huruf terakhir dari *head* graf, E. Sedangkan tabel pada bagian kanan atas Gambar 5.4 adalah tabel yang

menunjukkan penyimpanan elemen pada *container map*, dengan keterangan apabila berwarna hijau berarti elemen tersebut sedang digunakan, dan apabila berwarna merah berarti elemen tersebut telah dihapus dari *container map*. Pada Gambar tersebut elemen ATE berwarna hijau dan belum ada elemen yang dihapus.



Gambar 5.5. Perubahan Variabel pada Iterasi A ke-1 dan B ke-1 Uji Kasus 1

Gambar 5.5 adalah saat memasuki iterasi pertama pada *loop* B. Elemen ATE, yang merupakan *head* graf, telah dihapus dari *container map*. Yang digunakan sekarang adalah *key* elemen TER, karena K-1 huruf depannya sama dengan K-1 huruf belakang *key* elemen sebelumnya (ATE). Isi dari variabel *current* berubah menjadi ER, sedangkan isi dari variabel *result* ditambah huruf terakhir *key* elemen yang sedang digunakan (R), dari E menjadi ER.

N	10		**KEY**	**VALUE**
K	3		ATE	1
			ELO	1
Iterasi A ke -	1		ERM	1
Iterasi B ke -	2		LON	1
			MEL	1
Head Graf	ATE		RME	1
			TER	1
lastused	AT		WAT	1
current	RM			
result	ERM		Sedang digunakan	
word			Telah dihapus	

Gambar 5.6. Perubahan Variabel pada Iterasi A ke-1 dan B ke-2 Uji Kasus 1

Pada iterasi ke-2 *loop* B (Gambar 5.6), elemen TER dihapus, yang digunakan sekarang adalah *key* elemen ERM. Variabel *current* berubah menjadi RM dan variabel *result* menjadi ERM.

N	10		**KEY**	**VALUE**
K	3		ATE	1
			ELO	1
Iterasi A ke -	1		ERM	1
Iterasi B ke -	3		LON	1
			MEL	1
Head Graf	ATE		RME	1
			TER	1
lastused	AT		WAT	1
current	ME			
result	ERME		Sedang digunakan	
word			Telah dihapus	

Gambar 5.7. Perubahan Variabel pada Iterasi A ke-1 dan B ke-3 Uji Kasus 1

Pada iterasi ke-3 *loop* B (Gambar 5.7), elemen ERM dihapus, yang digunakan sekarang adalah *key* elemen RME. Variabel *current* berubah menjadi ME dan variabel *result* menjadi ERME.

			KEY	VALUE
N	10		ATE	1
K	3		ELO	1
			ERM	1
Iterasi A ke -	1		LON	1
Iterasi B ke -	4		MEL	1
			RME	1
Head Graf	ATE		TER	1
			WAT	1
lastused	AT			
current	EL			
result	ERMEL		Sedang digunakan	
word			Telah dihapus	

Gambar 5.8. Perubahan Variabel pada Iterasi A ke-1 dan B ke-4 Uji Kasus 1

Pada iterasi ke-4 *loop* B (Gambar 5.8), elemen RME dihapus, yang digunakan sekarang adalah *key* elemen MEL. Variabel *current* berubah menjadi EL dan variabel *result* menjadi ERMEL. Pada iterasi ke-5 *loop* B (Gambar 5.9), elemen MEL dihapus, yang digunakan sekarang adalah *key* elemen ELO. Variabel *current* berubah menjadi LO dan variabel *result* menjadi ERMELO.

N	10		**KEY**	**VALUE**
K	3		ATE	1
			ELO	1
Iterasi A ke -	1		ERM	1
Iterasi B ke -	5		LON	1
			MEL	1
Head Graf	ATE		RME	1
			TER	1
lastused	AT		WAT	1
current	LO			
result	ERMELO		Sedang digunakan	
word			Telah dihapus	

Gambar 5.9. Perubahan Variabel pada Iterasi A ke-1 dan B ke-5 Uji Kasus 1

N	10		**KEY**	**VALUE**
K	3		ATE	1
			ELO	1
Iterasi A ke -	1		ERM	1
Iterasi B ke -	6		LON	1
			MEL	1
Head Graf	ATE		RME	1
			TER	1
lastused	AT		WAT	1
current	ON			
result	ERMELON		Sedang digunakan	
word			Telah dihapus	

Gambar 5.10. Perubahan Variabel pada Iterasi A ke-1 dan B ke-6 Uji Kasus 1

Pada iterasi ke-6 *loop* B (Gambar 5.10), elemen ELO dihapus, yang digunakan sekarang adalah *key* elemen LON. Variabel *current* berubah menjadi ON dan variabel *result* menjadi ERMELON.

N	10	**KEY**	**VALUE**
K	3	ATE	1
		ELO	1
Iterasi A ke -	1	ERM	1
Iterasi B ke -	7	LON	1
		MEL	1
Head Graf	ATE	RME	1
		TER	1
lastused	AT	WAT	1
current	TE		
result	ERMELON	Sedang digunakan	
word	ERMELON	Telah dihapus	

Gambar 5.11. Perubahan Variabel pada Iterasi A ke-1 dan B ke-7 Uji Kasus 1

Pada iterasi ke-7 *loop* B (Gambar 5.11), elemen LON dihapus, dan akan keluar dari perulangan pada *loop* B. Variabel *current* tetap berisi TE dan variabel *result* tetap menjadi ERMELON. Isi variabel *word* disini berubah menjadi seperti isi variabel *result*, ERMELON, karena tidak ada lagi *key* elemen yang dapat terhubung dengan elemen sebelumnya (tidak ada *key* elemen yang mempunyai K-1 huruf depan sama dengan TE).

Saat memasuki iterasi ke-2 *loop* A (Gambar 5.12), head graf otomatis akan berubah menjadi *key* elemen pertama yang masih tersisa di dalam *container map*, WAT. Dengan begitu isi dari variabel *lastused* juga ikut berubah menjadi WA.

			KEY	VALUE
N	10		ATE	1
K	3		ELO	1
			ERM	1
Iterasi A ke -	2		LON	1
Iterasi B ke -	0		MEL	1
			RME	1
Head Graf	WAT		TER	1
			WAT	1
lastused	WA			
current	AT			
result	T		Sedang digunakan	
word	ERMELON		Telah dihapus	

Gambar 5.12. Perubahan Variabel pada Iterasi A ke-2 dan B ke-0 Uji Kasus 1

Isi dari variabel *current* berubah menjadi AT dan isi dari variabel *result* berubah menjadi T. Secara teknis isi dari variabel *result* telah dipindahkan ke variabel *word*, variabel *result* dikosongkan, kemudian diisi dengan huruf terakhir dari *key* elemen *head* graf.

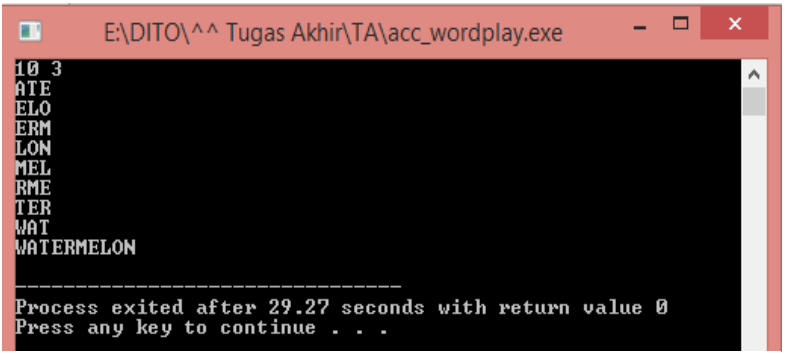
			KEY	VALUE
N	10		ATE	1
K	3		ELO	1
			ERM	1
Iterasi A ke -	2		LON	1
Iterasi B ke -	1		MEL	1
			RME	1
Head Graf	WAT		TER	1
			WAT	1
lastused	WA			
current	AT			
result	T		Sedang digunakan	
word	TERMELON		Telah dihapus	

Gambar 5.13. Perubahan Variabel pada Iterasi A ke-2 dan B ke-1 Uji Kasus 1

Pada iterasi ke-2 *loop* A dan iterasi ke-1 *loop* B (Gambar 5.13), elemen WAT dihapus, dengan begitu semua elemen sudah digunakan dan sekarang *container map* sudah kosong. Isi dari variabel *word* diubah lagi dengan menambahkan isi variabel *result* (T) ke depan isi variabel *word* (ERMELON), sekarang isi variabel *word* menjadi TERMELON. Setelah itu keluar dari *loop* B dan juga *loop* A, karena *container map* sudah kosong.

Pada proses terakhir, ubah lagi isi variabel *word* dengan menambahkan isi dari variabel *lastused* ke depan isi dari variabel *word*, WA ditambahkan ke depan TERMELON menjadi WATERMELON. Kemudian cetak isi variabel *word* sebagai *output* dari sistem.

Hasil output dari sistem ini berupa sebuah kata yang terbentuk dari susunan N-K+1 buah subkata yang diberikan sebagai input di awal sistem. Hasil yang didapatkan untuk uji kasus pertama ini dapat dilihat pada Gambar 5.14 berikut. Hasilnya sesuai dengan hasil dari visualisasi tiap proses pada sistem yang telah dilakukan, dan hasil yang didapatkan sesuai dengan harapan penulis.



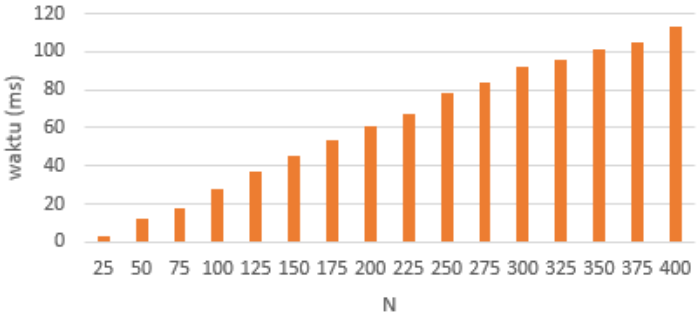
Gambar 5.14. Output Uji Kasus Pertama

5.2.2. Uji Coba Submission

Uji coba dilakukan dengan cara memberikan masukan dengan nilai N dan K yang berbeda-beda, dengan tujuan untuk melihat grafik waktu yang dibutuhkan untuk menjalankan program dengan input yang diberikan. Uji submission dilakukan dua kali. Uji submission pertama dilakukan dengan memberikan masukan nilai K statis, K=3, dan nilai N dinamis, N=25,50,75,...,400. Untuk uji submission kedua dilakukan dengan nilai N dinamis seperti pada uji submission pertama, akan tetapi nilai K statis yang digunakan adalah K=15. Masukan angka serta huruf untuk subkata tersebut diambil secara acak.

5.2.2.1. Uji Coba Submission Pertama

Nilai yang digunakan untuk uji coba submission pertama adalah K=3 dan N=25,50,75,...,400. Gambar 5.15 menunjukkan grafik hasil uji coba submission terhadap waktu yang digunakan (dalam mili sekon), tabel perubahan nilai N, K dan perubahan waktu ditunjukkan oleh Tabel 5.1.



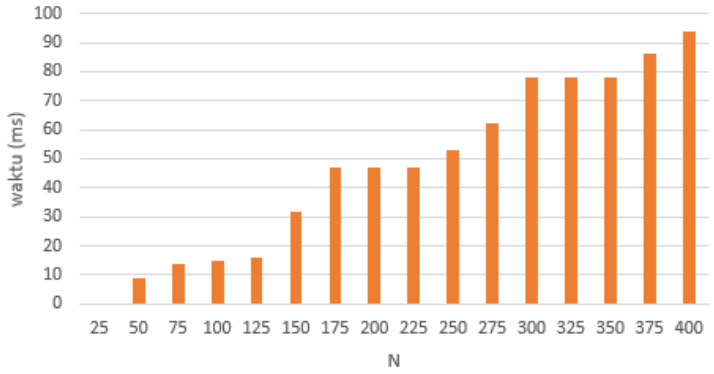
Gambar 5.15. Grafik N terhadap waktu dalam mili sekon pada Uji Submission Pertama

Tabel 5.1. Tabel Nilai N, K dan Waktu Running dalam mili sekon pada Uji Submission Pertama

K	N	Waktu(ms)
3	25	3
3	50	12
3	75	18
3	100	28
3	125	37
3	150	45
3	175	54
3	200	61
3	225	67
3	250	78
3	275	84
3	300	92
3	325	96
3	350	101
3	375	105
3	400	113

5.2.2.2. Uji Coba Submission Kedua

Nilai yang digunakan untuk uji coba submission yang kedua adalah K=15 dan nilai N sama seperti pada uji coba pertama, yaitu N=25,50,75,...,400. Gambar 5.16 menunjukkan grafik hasil uji coba submission terhadap waktu yang digunakan (dalam mili sekon), tabel perubahan nilai N, K dan perubahan waktu ditunjukkan oleh Tabel 5.2.



Gambar 5.16. Grafik N terhadap waktu dalam mili sekon pada Uji Submission Kedua

Tabel 5.2. Tabel Nilai N, K dan Waktu Running dalam mili sekon pada Uji Submission Kedua

K	N	waktu(ms)
15	25	0
15	50	9
15	75	14
15	100	15
15	125	16
15	150	32
15	175	47
15	200	47
15	225	47
15	250	53
15	275	62
15	300	78
15	325	78
15	350	78
15	375	86
15	400	94

Halaman ini sengaja dikosongkan

BAB VI
KESIMPULAN

Pada bab ini dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan dan saran mengenai hal-hal yang masih dapat dikembangkan dari Tugas Akhir ini.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap implementasi penyelesaian permasalahan *Word Play* dapat diambil kesimpulan sebagai berikut.

1. Implementasi graf hamilton dan penggunaan *Standard Template Library* (STL) dapat menyelesaikan permasalahan *Word Play* dengan benar.
2. Kompleksitas waktu yang dibutuhkan untuk seluruh proses adalah $O((N-K+1)log_2(N-K+1))$.

6.2 Saran

Saran yang diberikan dalam pengembangan algoritma penyelesaian masalah *Word Play* adalah melakukan optimasi *memory* agar didapatkan penggunaan *memory* yang lebih efisien.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] “Definisi Graf,” [Online]. Available: http://www.academia.edu/8723643/Definisi_Graf. [Diakses 3 Juni 2018].

[2] D. Wirdasari, TEORI GRAPH DAN IMPLEMENTASINYA, 2011.

[3] “Hamiltonian Path,” 4 Maret 2018. [Online]. Available: https://en.wikipedia.org/wiki/Hamiltonian_path. [Diakses 3 Juni 2018].

[4] “Geeksforgeeks,” [Online]. Available: https://www.geeksforgeeks.org/the-c-standard-template-library-stl/. [Diakses 27 Juni 2018].

[5] “std::map,” 12 April 2017. [Online]. Available: http://en.cppreference.com/w/cpp/container/map. [Diakses 3 Juni 2018].

[6] S. R. Labs, “WORD-WordPlay,” 19 April 2011. [Online]. Available: http://spoj.com/problems/WORD/. [Diakses 3 Juni 2018].

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Pramudito Hapriarso, lahir pada tanggal 19 April 1996 di Pati, Jawa Tengah. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya. Penulis memiliki hobi di bidang editing, olahraga dan travelling menjelajah tempat baru. Penulis memiliki beberapa pengalaman dalam organisasi seperti menjabat sebagai Staff Departemen Minat dan Bakat Himpunan Mahasiswa Teknik Computer-Informatika (HMTC) ITS periode 2015-2016 dan Ketua Dewan Perwakilan Angkatan (DPA) Teknik Informatika ITS periode 2016-2017. Penulis juga memiliki beberapa pengalaman dalam hal event organizing seperti menjadi staff Dana Usaha Big Event Schematics periode 2014-2015, staff ahli Reeva Big Event Schematics periode 2015-2016 dan juga sebagai staff Wahana Kreatif ITS EXPO 2015. Penulis dapat dihubungi melalui surel di pramuditoh1996@gmail.com.