Aplikasi Pohon untuk Code Completion List pada Visual Studio Code

Fitrah Ramadhani Nugroho - 13520030[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13520030@itb.ac.id

***Abstract*—Semakin berkembangnya dunia IT mengakibatkan banyaknya programmer sehingga diperlukannya aplikasi yang mampu membantu programmer mengetik kode dengan cepat yang berupa *code completion list*. *Code completion list* memberikan rekomendasi variabel, *keywords,* atau fungsi yang sering dipakai oleh pengguna sehingga mempercepat penulisan *code.* Implementasi dari *code completion list* dapat dilakukan dengan menggunakan Trie yang merupakan modifikasi dari struktur data Pohon.**

***Keywords*—Pohon, Trie, Hash, Python, Completion, List. Rekomendasi**

I. PENDAHULUAN

Pada zaman sekarang yang merupakan Revolusi Industri 4.0 ,bidang IT dan komputer semakin banyak diminati oleh banyak orang. Pekerjaan seperti programmer, software engineer, data scientist, game developer, dan masih banyak lagi semakin dicari oleh perusahaan. Pekerjaan tersebut sering berhubungan dengan menulis program atau menulis code.

Programmer atau sebutan untuk orang yang membuat program, menulis code dengan menggunakan bahasa pemrograman. Banyak sekali bahasa pemrograman yang bisa digunakan oleh *programmer* atau *coder.* Bahasa pemrograman yang banyak digunakan yaitu Python, Java, C, C++ dan C#. Agar code yang ditulis dapat dibuat menjadi program, diperlukan untuk meng-*compile* code yang ada menjadi program dengan menggunakan *compiler.*

Ada banyak sekali jenis compiler, ada compiler yang khusus untuk satu bahasa pemrograman yang disebut sebagai IDE seperti Dev C++, GCC yang hanya bisa meng-*compiler* bahasa C,C++, dan C#, ada juga *compiler* daring yang mampu meng-*compiler* berbagai macam bahasa seperti ideone.com, dan ada juga aplikasi yang bisa ditambah *extensions* sehingga mampu meng-*compiler* berbagai macam bahasa pemrograman hanya dengan men-*instal extensions* yang sesuai dengan bahasa tersebut.

Aplikasi yang mampu untuk meng-*compile* berbagai macam bahasa pemrograman disebut sebagai *source code editor*. *Source code editor* merupakan aplikasi *text editor* yang khusus untuk menulis code atau program. Dengan menggunakan *source code editor, Coder* atau *programmer* tidak perlu repot-repot menggunakan aplikasi yang berbeda ketika ingin menulis code dengan bahasa pemrograman yang berbeda. Contoh dari *source code editor* yaitu Eclipse, Sublime Text, Notepad++, dan Visual Studio Code.

Visual Studio Code (VSC) merupakan *source code editor* yang paling populer dan banyak dipakai oleh *programmer.* Kepopuleran dari VSC disebabkan oleh mudahnya menulis code di VSC karena VSC mengkomodasi berbagai macam bahasa pemrograman hanya dengan meng-*install extension* yang sesuai. Selain banyak *extension* di VSC yang bisa memudahkan *programmer* atau *coder* dalam menulis code. Salah satu *extension* yang memudahkan programmer yaitu Pylance dan Visual Studio Intellicode.

Pylance adalah *extension* yang berfungsi untuk mempermudah menulis code dengan Python. Fitur-fitur dari Pylance yaitu *code completion, type checking, paramater suggestion,* dll. Sedangkan Visual Studio Intellicod adalah *extension* yang memberikan rekomendasi *completion list* berbasis AI. Dengan menggunakan salah satu dari kedua *extension* diatas, *extension* akan memberikan rekomendasi *code completion list* pada sebuah kata sehingga programmer tidak perlu repot repot menulis kata tersebut dengan utuh.

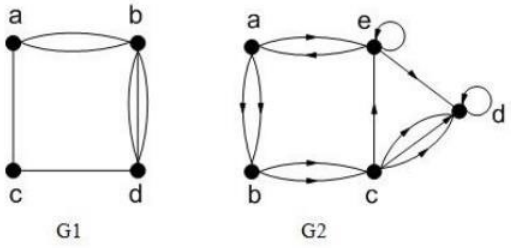
Pada makalah ini penulis akan membahas mengenai aplikasi dari Pohon terutama *Trie* untuk *code completion list* pada kedua *extension* yaitu Pylance dan Visual Studio Intellicode.

II. LANDASAN TEORI

A. Graf

Graf adalah himpunan tidak kosong dari simpul-simpul (*vertices*) dengan himpunan sisi (*edges*) yang menghubungkan sepasang simpul. Notasi graf yaitu Graf G = (V, E) dengan V = {*v1, v2, ..., vn*} dan E = {*e1, e2, ..., en*}. Graf sering digunakan untuk merepresentasikan hubungan diantara objek-objek yang ada.

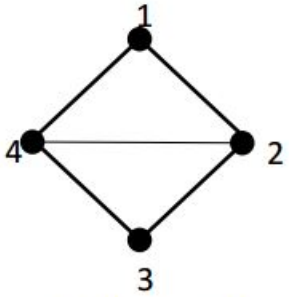
Graf terdiri dari dua jenis yaitu graf berarah dan graf tak berarah. Graf tak berarah terdiri dari graf sederhana dan graf tak sederhana. Graf sederhana yaitu graf yang tidak mengandung sisi ganda atau sisi gelang. Graf tak sederhana dibedakan lagi menjadi dua yaitu graf ganda adalah graf yang mengandung sisi ganda dan graf semu adalah graf yang mengandung sisi gelang. Sedangkan graf berarah dibedakan menjadi dua graf berarah sederhana dan graf ganda berarah. Graf berarah adalah graf yang setiap sisinya memiliki arah menuju simpul.



Gambar 2.1 G1 graf berarah dan G2 graf tidak berarah (*sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf*)

Graf memiliki sebelas terminologi yaitu:

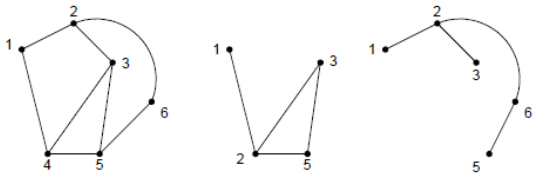
1. Ketetanggan (*Adjacent*)



Gambar 2.2 Contoh graf bertetangga (*sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf*)

Jika kedua simpul terhubung langsung maka dua buah simpul dikatakan bertetangga. Berdasarkan graf pada gambar 2.2, simpul 1 bertetangga dengan simpul 2 karena ada sisi yang terhubung di simpul 1 dan 2. Bertetanggaan bisa disimbolkan dengan sisi e = (1, 2) dengan 1 adalah simpul 1 dan 2 adalah simpul 2.

2. Bersisian (*incidency*)
   Sebuah sisi e = (A, B) dikatakan bersisian dengan simpul A dan B jika sisi tersebut terhubung dengan simpul A dan B. Sebagai contoh pada graf di gambar 2.2 sisi (3, 4) bersisian dengan simpul 3 dan 4.
3. Derajat (*Degree*)
   Derajat pada suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut. Notasi dari derajat adalah d(v) dengan v adalah simpul. Contohnya pada graf di gambar 2.2, derajat dari simpul 4 atau d(4) = 3 adalah 3 karena simpul 4 bersisian dengan 3 sisi.
4. Lintasan (Path)
   Lintasan yaitu barisan dari simpul-simpul dan sisi-sisi yang saling berhubungan. Lintasan dengan panjang n dengan simpul awal V0 ke simpul tujuan Vn terdiri dari e1 = (V0,V1), e2 (V1,V2), ..., en (Vn-1,Vn). Misalnya pada graf di gambar 2.2 lintasan 3, 4, 2, 1 adalah lintasan dengan barisan sisi (3,4), (4,2), (2,1) dengan panjang lintasan sebesar 3.
5. Siklus (*cycle*) atau Sirkuit (*circuit*)
   Siklus atau sirkuit adalah lintasan yang berawal dan berakhir pada simpul yang sama. Contohnya pada graf di gambar 2.2 yaitu 3, 4, 2, 3 adalah sebuah sirkuit dengan panjang 3.
6. Keterhubungan (*connected*)
   Sebuah graf disebut graf terhubung (*connected graph*) jika untuk setiap pasang simpul Vi dan Vj terdapat lintasan dari Vi ke Vj. Graf pada gambar 2.2 dan 2.1 disebut graf terhubung karena tiap simpulnya terhubung satu sama lain.
7. Upagraf (*Subgraph*) dan Komplemen Upagraf
   Upagraf adalah bagian dari graf yang merupakan *subset* dari sebuah graf. Sedangkan komplemen dari upagraf G1 terhadap graf G adalah graf G2 = (V2,E2) sehingga E2 = E-E1 dan V2 adalah himpunan simpul yang anggota-anggota E2 bersisian dengannya.

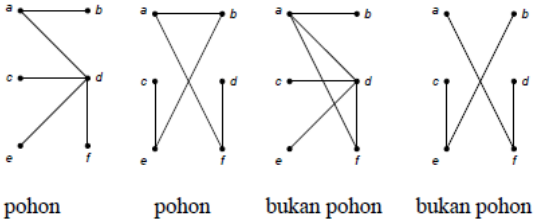


Gambar 2.3 Graf dikiri berupa Graf G1 dengan graf ditengah adalah upagraf G1 dan graf di kanan adalah komplemen dari upagraf di tengah (*sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf*)

8. Upagraf Merentang (*Spanning Subgraph*)
   Sebuah upagraf dikatakan upagraf rentang jika upagraf tersebut mengandung semua simpul dari graf.
9. *Cut-Set*
   Cut-set dari graf terhubung G adalah himpunan sisi yang jika dibuang menyebabkan G tidak terhubung. Oleh karena itu, *cut-set* selalu menghasilkan dua buah komponen.
10. Graf Berbobot (*Weighted Graph*)
    Graf berbobot adalah graf yang tiap sisinya terdapat sebuah nilai atau bobot.

B. Pohon

Pohon adalah graf tak berarah terhubung yang tidak mengandung sirkuit. Kumpulan dari pohon yang saling lepas disebut sebagai hutan (*forest*). Selain itu, hutan adalah graf tidak terhubung yang tidak mengandung sirkuit dengan setiap komponen di dalam graf terhubung berupa pohon.



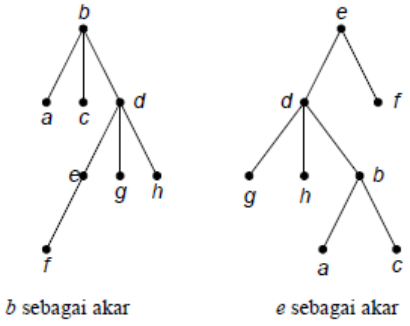
Gambar 2.4 Contoh Pohon dan Bukan Pohon (*sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf*)

Jika G = (V, E) adalah graf tak-berarah sederhana dan jumlah simpulnya n. Pohon dapat dinyatakan sebagai berikut:

- G adalah pohon
- Setiap pasang simpul di G terhubung dengan lintasan tunggal
- G adalah graf terhubung dan memiliki m = n-1 sisi
- Graf G tidak mengandung sirkuit
- Semua sisi dari Graf G adalah jembatan.

Salah satu jenis pohon yaitu pohon merentang. Pohon merentang dari graf terhubung adalah upagraf merentang yang berupa pohon. Jika sebuah sirkuit di dalam graf dipotong maka akan diperoleh pohon merentang. Setidaknya terdapat satu buah pohon merentang di setiap graf terhubung. Graf tak terhubung dengan n komponononen mempunyai n buah hutan merentang yang disebut hutan merentang (*spanning tree*). Pohon merentang yang dihasilkan dari graft berbobot dengan bobot minimum disebut pohon merentang minimum (*minimum spanning tree*).

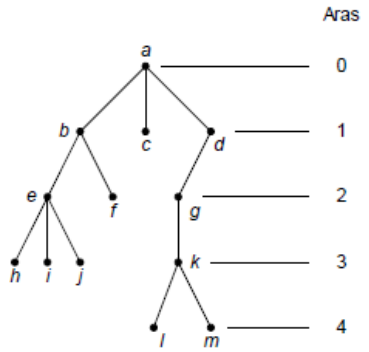
Jenis pohon lainnya yaitu pohon berakart (*rooted tree*), yaitu pohon yang satu buah simpulnya dibuat sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah.



Gambar 2.5 Contoh Pohon berakar dengan Akar yang Berbeda
(*sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag2.pdf*)

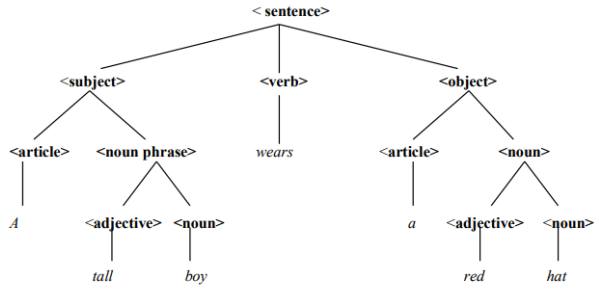
Pohon berakar memiliki beberapa terminologi yaitu :

1. Anak (*child / children*) dan Orangtua *(parent)*
   Pada pohon sebelah kiri di gambar 2.5 a, c, dan d merupakan anak-anak simpul b dengan b adalah orangtua dari anak-anak itu.
2. Lintasan (path)
   Lintasan dari *b* ke *h* di pohon sebelah kiri di gambar 2.5 adalah *b, d, h* dengan panajng lintasan adalah 2.
3. Saudara kandung (*sibling*)
   Pada pohon sebelah kiri di gambar 2.5 *a* adalah saudara kandung *c* dan *d.* Tetapi *a* bukan saudara kandung dari *e* karena orang tua kedua anak berbeda.
4. Upapohon (*subtree*)
   Upapohon dari suatu pohon adalah bagian dari pohon berakar yang memiliki simpul minimal satu.
5. Derajat (*degree*)
   Berbeda dengan graf, derajat pada pohon berakar adalah jumlah anak atau jumlah upapohon pada simpul tersebut. Sebagai contoh pada pohon sebelah kiri di gambar 2.5 derajat dari simpul *d* adalah 3 karena *d* memiliki 3 anak.
6. Daun (*leaf*)
   Daun adalah simpul yang berderajat nol atau tidak memiliki anak. Pada pohon sebelah kiri di gambar 2.5 simpul *a, c, f, g*, dan *h* adalah simpul daun.
7. Simpul Dalam (*internal nodes*)
   Simpul dalam adalah kebalikan dari daun yaitu simpul yang mempunyai anak. Pada pohon sebelah kiri di gambar 2.5 simpul b, e, dan d adalah simpul dalam.
8. Aras (*level*) atau Tingkat
   Tingkat atau aras adalah kedalaman sebuh simpul dari simpul akar. Pada gambar dibawah simpul *b* memiliki aras sebesar 1 karena simpul *b* memiliki panjang lintasan sebesar 1 dari akar.



Gambar 2.6 Contoh Pohon berakar dengan Aras
(*sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag2.pdf*)

9. Tinggi (*height*) atau Kedalaman (*depth*)
   Tinggi atau kedalaman dari suatu pahan adalah aras atau tingkat maksimum dari pohon tersebut. Pohon di gambar 2.6 mempunyai tinggi 4.
10. Pohon n-ary
    Pohon n-ary adalah pohon berakar yang setiap simpul cabangnya mempunyai paling banyak n buah anak. Jika setiap simpul cabangnya mempunyai tepat n anak maka pohon n-ary dikatakan teratur atau penuh (*full*).



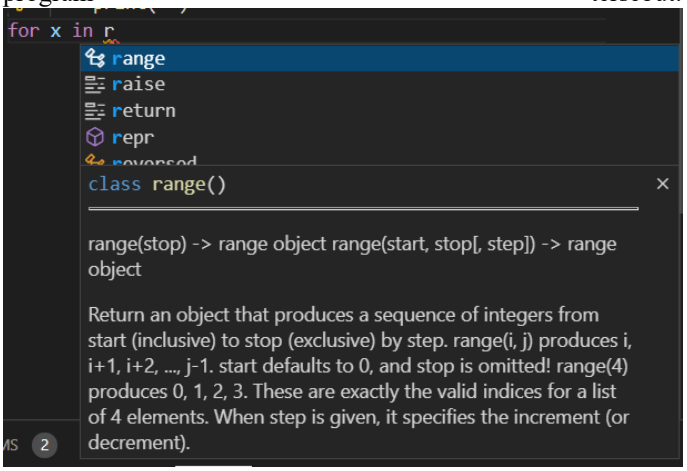
Gambar 2.6 Contoh Pohon n-ary dengan n adalah 3
(*sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag2.pdf*)

III. PEMBAHASAN

A. Fitur Completion List pada Pylance dan Visual Studio Intellicode.

Pylance sebagai *extension* yang berfungsi untuk memberikan dukungan pada bahasa Python berupa fitur-fitur seperti Docstrings, paramater suggestions, code outline, Code completion dan lain-lainnya. Fitur yang akan dibahas pada makalah ini yaitu fitur code completion. Fitur code completion memberikan daftar saran kepada penulis code untuk mempersingkat penulisan code karena fitur code completion

berfungsi untuk mempercepat dan mempermudah menulis code berdasarkan *library* pada bahasa Python. Selain memberikan daftar rekomendasi *code completion* berdasarkan fungsi-fungsi yang ada di *library* Python, Pylance juga memberikan rekomendasi berdasarkan variabel yang telah di deklarasi oleh penulis code. Fitur *code completion list* tidak hanya memberikan daftar rekomendasi yang ada tetapi juga mengurutkannya berdasarkan seberapa sering fungsi itu dipakai dan konteks pada program tersebut.



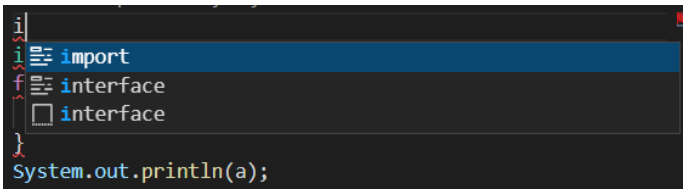
Gambar 3.1 Contoh Penggunaan *Code Completion List*

Pada gambar 3.1 terlihat bahwa ketika pengguna mengetik huruf ‘r’ akan muncul rekomendasi mengenai *keyword* dari Python dan fungsi yang sering dipakai. Kata “range” menempati posisi teratas pada *completion list* diikuti dengan kata “raise”, “return”, “repr”, dan masih ada lagi dibawahnya. Kata “range” merupakan fungsi yang sering digunakan ketika menggunakan *keywords* iterasi yaitu “for”. Sedangkan kata “raise” dan “return” adalah *keyword* di Python.

Gambar 3.2 Contoh Penggunaan *Code Completion List*

Selain itu, fitur *code completion list* juga memiliki fitur *spelling checker* yang mana akan memberikan daftar rekomendasi ketika sebuah kata pada Python bersifat unik yang tidak ada di *library* atau belum pernah di deklarasi sebagai variabel. Pada gambar 3.2 terlihat ketika pengguna mengetik kata “rge” *code completion* akan memberikan rekomendasi kata yaitu “range” karena huruf ‘r’, ‘g’, dan ‘e’ yang ada pada kata “rge” ada juga di kata “range”.Figure axis labels are often a source of confusion.

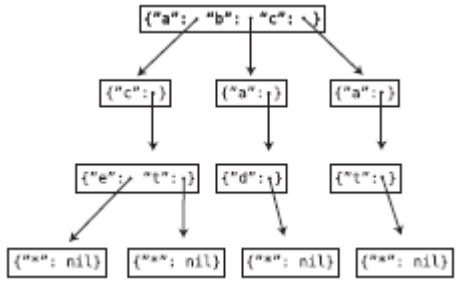
Selain Pylance, *extension* lainnya yang memiliki fitur *code completion list* adalah Visual Studio Intellicode. Berbeda dengan Pylance, Visual Studio Code bisa digunakan pada bahasa selain Python yaitu Java dan JavaScript.



Gambar 3.3 Contoh Penggunaan *Code Completion List pada Java*

B. Latar Belakang Trie

Banyak struktur data yang dapat digunakan sebagai implementasi dari *code completion list.* Namun saat ini kita hanya membahas struktur data berbasis pohon yaitu Trie sebagai aplikasi pohon dalam *code completion list*. Trie adalah pohon n-ary yang sering digunakan sebagai fitur berbasis teks salah satunya yaitu *autocomplete.* Hal ini disebabkan struktur Trie yang tidak hanya berupa pohon namun simpul dari pohon tersebut berupa *hash table* sehingga mempermudah untuk menyimpan huruf dan kata.

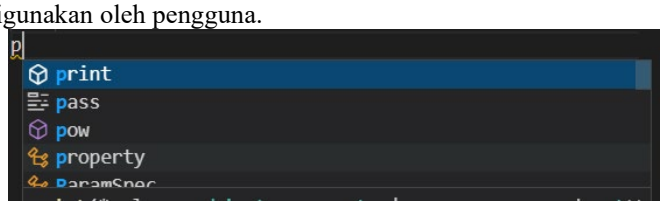


Gambar 3.4 Contoh Pohon n-ary dengan n adalah 3 (*sumber: A Common-Sense Guide to Data Structures and Algorithms, Second Edition by Jay Wengrow.pdf*)

Contoh dari implementasi Trie terlihat pada gambar 3.4, simpul akar adalah *hash table* yang terisi oleh huruf ‘a’,’b’, dan ‘c’. Setiap huruf di *hash table* pada simpul juga memiliki pohon masing masing. Huruf ‘a’ memiliki anak simpul dengan huruf ‘c’ dan simpul huruf ‘c’ juga memiliki anak simpul dengan huruf ‘e’ dan ‘t’ yang masing-masing membentuk kata “ace” dan “act. Pada akhir setiap kata diikuti dengan simpul berisi karakter ‘*’ untuk menandai terbentuknya kata. Aplikasi Trie bermacam-macam mulai dari *autocomplete* hingga *spelling suggestion*. Namun kali akan dibahas implementasi Trie sebagai *code completion list* di Visual Studio Code

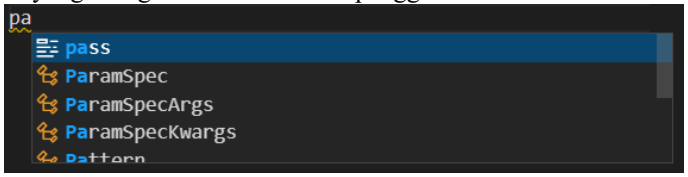
C. Implementasi Trie sebagai Code Completion List

Untuk memberikan daftar rekomendasi, Pylance membaca masing-masing karakter dari pengguna kemudian menampilkan prediksi fungsi, *keywords* atau variabel yang mungkin digunakan oleh pengguna.



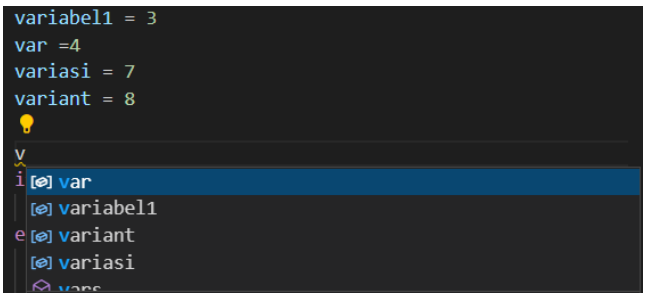
Gambar 3.5 Contoh daftar rekomendasi awal

Jika pengguna memasukan karakter yang berbeda dari daftar rekomendasi yang ada, maka akan muncul daftar rekomendasi baru yang mungkin dari masukkan pengguna.



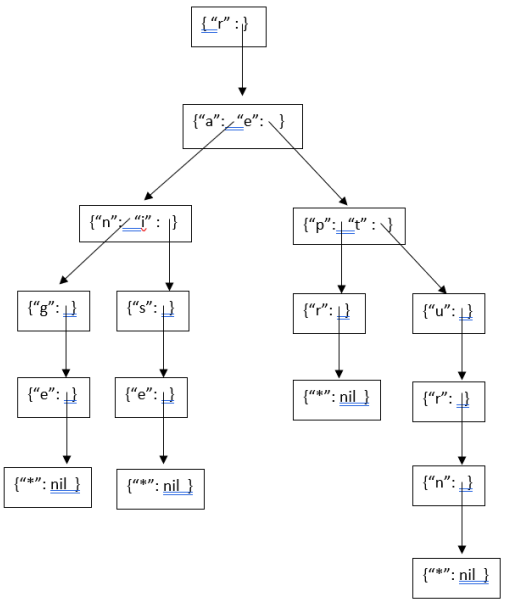
Gambar 3.6 Contoh daftar rekomendasi lebih lanjut

Seperti yang terlihat dari gambar 3.5 dan gambar 3.6 daftar rekomendasi berubah menyesuakan masukkan pengguna.



Gambar 3.7 Contoh daftar rekomendasi variabel

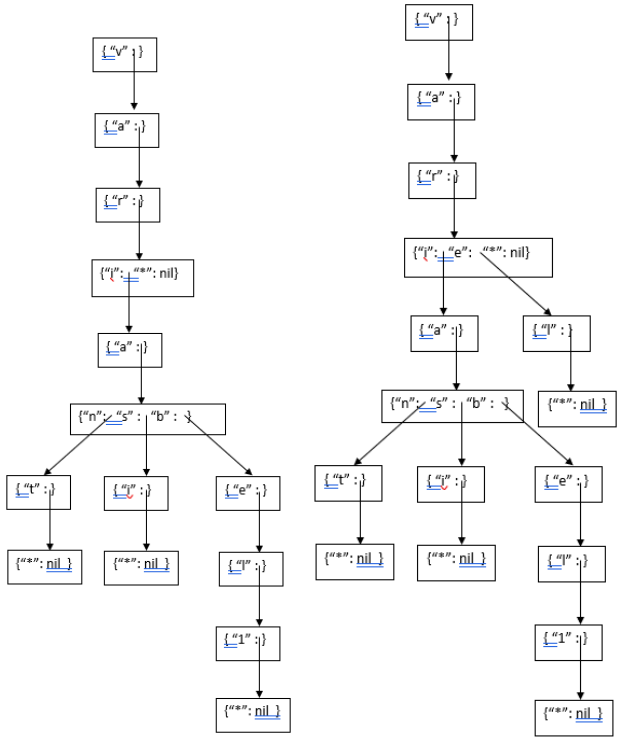
Pylance juga memberikan rekomendasi variabel yang sudah di deklarasi sebelumnya seperti yang ditunjukkan gambar 3.7. Terlihat bahwa urutan daftar rekomendasi terurut dari urutan variabel itu di deklarasi.



Gambar 3.8 Aplikasi Trie dari *code completion list* gambar 3.1

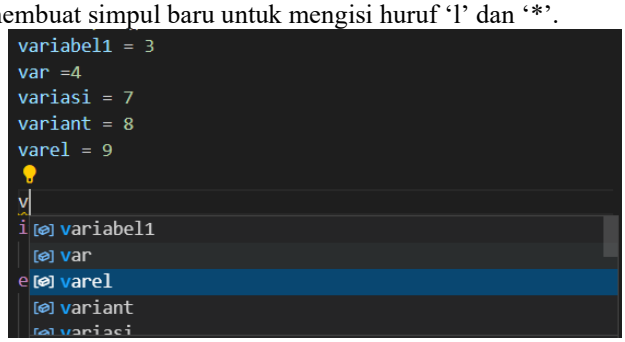
Jika hasil dari daftar rekomendasi pada gambar 3.1 dibuat menjadi trie maka akan seperti pada gambar 3.8. Terlihat bahwa "r" adalah akar dari trie tersebut yang merupakan huruf awal yang diketikkan pengguna. Kemudian simpul dari akar menunjuk ke simpul yang berisi kata "a" dan "e" yang merupakan kemungkinan yang ada. Dari simpul itu sendiri bercabang menjadi dua simpul dimana setiap simpul terdiri dari dua huruf yang berbeda. Huruf pada simpul juga menunjuk ke simpul berikutnya hingga simpul terakhir yang berisi asterisk atau '*' yang menandai berakhirnya pencarian. Berdasarkan Trie yang ada ini Pylance mencari kemungkinan yang ada berdasrkan Trie diatas dan menaruh setiap hasilnya pada *list* atau daftar. *List* inilah yang ditampilkan pada Python. Pengguna memilih hasil pada hasil tersebut dengan mencari kata yang diinginkan lalu klik *Enter.* Namun pengguna juga bisa tidak menggunakan list yang tersedia jika ingin mendeklarasi variabel baru atau menulis fungsi yang tidak ada sebelumnya.

Lalu bagaimana ketik pengguna ingin mendeklarasi variabel baru. Seperti yang terlihat di gambar 3.8, Pylance juga memasukkan variabel yang di deklarasikan pengguna ke dalam *code completion list* sehingga pengguna tidak perlu mengetik lagi variabel yang sudah di deklarasi. Pylance memasukkan variabel ke dalam trie dengan mencocokkan huruf-huruf di variabel ke dalam data-base Trie.



Gambar 3.9 Aplikasi Trie dari *code completion list* gambar 3.7 (kiri), Pohon Trie setelah ditambah variabel baru "varel" (kanan).

Gambar diatas yaitu perubahan pohon sebelum ditambah dideklarasi variabel baru "varel" dengan sesudah dideklarasi variabel baru. Terlihat bahwa Pylance berusaha mencocokan terlebih dahulu huruf-huruf dari "varel" dengan huruf di pohon Trie. Huruf 'v', 'a', dan 'r' sudah ada di Trie sehingga tidak perlu menambah simpul terbaru. Namun huruf 'e' tidak ada di *hash table* pada huruf 'i' sehingga Pylance menambahkkan huruf 'e' ke *hash table* tersebut. Setelah itu Pylance juga membuat simpul baru untuk mengisi huruf 'l' dan '*'.



Gambar 3.10 Contoh daftar rekomendasi variabel setelah penambahan variabel "varel"

Terlihat di gambar 3.10 bahwa setelah di deklarasikan variabel baru yaitu "varel", variabel tersebut ada di *code completion list*.

Namun *code completion list* di gambar 3.10 masih terurut secara acak. Hal itu disebabkan tiap variabel yang baru dipakai satu kali yaitu ketika deklarasi variabel. Sesuai yang disebutkan sebelum Pylance mengurutkan *code completion* berdasarkan seberapa sering fungsi atau variabel itu digunakan. Bagaimana cara Trie membuat *code completion list* menjadi terurut berdasarkan frekuensi penggunaan variabel, *keyword* atau

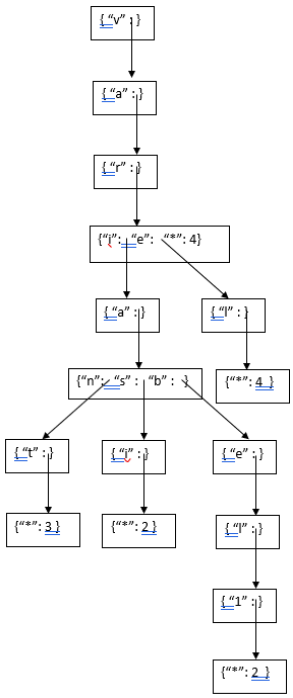
fungsi. Cara yang dipakai agar Trie bisa mempresentasikan frekuensi penggunaan variabel atau fungsi yaitu dengan cara menambahkan angka pada setiap asterisk atau "*". Angka ini melambangkan seberapa sering kata tersebut digunakan oleh pengguna. Karena asterisk melambangkan akhir dari setiap kata, dengan menambahkan angka pada asterisk, Trie hanya perlu melihat besar angka di asterisk tersebut. Jika sebuah angka lebih kecil dari nilai di *code completion list* maka kata yang direpresntasikan oleh angka tersebut akan ditaruh dibelakang *list*. Sedangkan jika angka lebih besar dari nilai yang ada di *code completion list* maka kata akan ditaruh di depan *list.* Jika angka berada di antara nilai di *code completion list* maka kata akan diurutkan pada *list* dari nilai yang paling besar ke nilai yang paling kecil.

```
variabel1 = 3
var =4
variasi = 7
variant = 8
varel = 9

varel = variabel1 + 1
var = var *7 *var
for x in range (variasi):
    variant = variant +1
    varel = varel **2
```

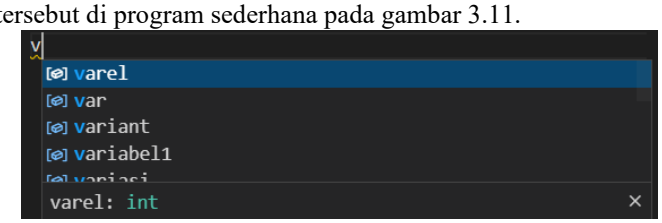
Gambar 3.11 Contoh Program Sederhana

Berdasarkan program sederhana pada gambar 3.11 kita akan mencoba menghitung kemunculan tiap variabel dan mencoba mengaplikasikannya ke dalam Trie yang sudah dimodifikasi. Pertama variabel "variabel1" muncul sebanyak dua kali. Kemunculan pertama saat deklarasi variabel dan kedua saat persamaan "varel = variabel1 +1". Untuk variabel lainnya yaitu variabel "var" muncul sebanyak empat kali, variabel "variasi" muncul sebanyak dua kali, variabel "variant" muncul sebanyak tiga kali, variabel "varel" muncul sebanyak empat kali. Jika angka tersebut diaplikasikan kepada Trie di gambar 3.9 maka akan menjadi :



Gambar 3.12 Aplikasi Trie sesuai Program Sederhana pada gambar 3.11

Gambar 3.12 merupakan modifikasi Trie dari gambar 3.9 dengan menggunakan program sederhana dari gambar 3.11 untuk menghitung frekuensi. Pada gambar 3.12 terlihat perubahan pada nilai asterisk yang sebelumnya berisikan "nil" sekarang bernilai angka sesuai dengan kemunculan variabel tersebut di program sederhana pada gambar 3.11.



Gambar 3.13 Daftar rekomendasi sesuai Program Sederhana pada gambar 3.11

Gambar 3.13 menunjukkan perubahan *code completion list* yang sebelumnya seperti gambar 3.10. Pada gambar 3.10 posisi teratas pada *code completion list* adalah variabel "variabel1". Namun pada gambar 3.13 posisi teratas berubah menjadi variabel "varel". Hal itu karena "varel" muncul empat kali sehingga kata tersebut menempati posisi teratas. Kemudian di posisi kedua adalah variabel "var" dengan nilai frekuensi sama seperti "varel" yaitu empat. Posisi ketiga yaitu "variant" dengan frekuensi adalah tiga. Posisi keempat dan terakhir yaitu "variabel1" dan "variasi" yang memiliki nilai frekuensi sama sama satu.

Berdasarkan fitur-fitur yang dijelaskan diatas menunjukkan bahwa Trie bisa diimplementasikan sebagai *code completion list.* Namun Trie adalah salah satu cara dari implementasi *code completion list*. Masih banyak struktur data lain yang bisa mengimplementasikan *code completion list* dengan lebih efisien.

IV. KESIMPULAN

Pohon memiliki banyak sekali kegunaan salah satunya sebagai *code completion list* dengan menggunakan Trie yang merupakan modifikasi dari Pohon. Berdasarkan penjelasan diatas Trie dapat diimplementasikan menjadi *code completion list* dengan cara membaca masukkan pengguna lalu memeriksa di Trie dan menampilkan rekomendasi yang sesuai dengan masukkan pengguna. Keberadaan. Keberadaan *code completion list* sangat membantu programmer atau coder karena mengehemat tenaga dan mempersingkat waktu dalam mengetik program.

V. UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur dan rasa terima kasih kepada Allah SWT atas rahmat dan pertolongan-Nya, penulis dapat menyelesaikan makalah ini. Selain itu, ucapan terima kasih penulis panjatkan kepada orang tua penulis karena telah mendukung penulisan makalah ini. Terakhir penulis ucapkan terima kasih kepada Bapak Rinaldi Munir sebagai dosen pengajar mata kuliah Matematika Diskrit K1 yang telah menyediakan waktu dan tenaga beliau untuk mengajarkan ilmu Matematika Diskrit. Semoga ilmu yang diajarkan beliau berguna di masa depan. Penulis berharap makalah ini dapat menjadi pembelajaran bagi pembaca dan dimanfaatkan dengan baik.

VI. DAFTAR PUSTAKA

[1] Munir, Rinaldi. 2014. *Matematika Diskrit*. Bandung: Informatika Bandung.

[2] Wengrow, Jay. 2020. *A Common-Sense Guide to Data Structures and Algorithms, Second Edition (Book style).* Raleigh, NC: Pragmatic Bookshelf.

[3] Scnur, Bill, dkk. *Pylancce.* https://github.com/microsoft/pylance-release, diakses pada tanggal 11 Desember 2021.

[4] Thomas, Mark, dkk. *Visual Studio Intellicode.* https://visualstudio.microsoft.com/services/intellicode/ , diakses pada tanggal 11 Desember 2021.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 13 Desember 2021

Fitrah Ramadhani Nugroho
13520030