Penerapan Algoritma *Greedy* dalam Pencarian Rantai Penjumlahan Terpendek

Irwan Kurniawan – 135 06 090

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl Ganesha 10, Bandung
e-mail: if16090@students.if.itb.ac.id

ABSTRAK

Rantai Penjumlahan adalah sebuah masalah yang telah ada sejak lama. Masalah yang menarik dari rantai penjumlahan ini adalah cara untuk menemukan suatu urutan rantai penjumlahan yang memiliki urutan terpendek (rantai penjumlahan terpendek). Telah banyak pakar matematika dan ilmu komputer mencoba untuk menerapkan strategi yang tepat dalam memecahkan masalah pencarian rantai penjumlahan terpendek.

Ide dari rantai penjumlahan ini cukup sederhana. Kita harus mencari urutan angka – angka dimulai dari angka 1 hingga angka ke – n, dimana angka ke – n adalah akhir dari rantai penjumlahan. Ada beberapa aturan dalam urutan angka – angka rantai penjumlahan ini. Aturan pertama, Setiap angka – angka yang berada di dalam urutan, kecuali angka pertama, haruslah angka – angka yang berasal dari penjumlahan angka – angka sebelumnya (mungkin sama). Aturan kedua, angka – angka yang berada di dalam urutan harus terurut menaik, atau dengan kata lain, angka – angka yang berada di dalam urutan rantai penjumlahan ini, harus memiliki nilai yang lebih besar dibandingkan angka sebelumnya.

Makalah ini bertujuan untuk menjelaskan salah satu penerapan strategi algoritmik dalam mencari solusi rantai penjumlahan terpendek. Strategi yang digunakan oleh penulis di dalam makalah ini adalah algoritma *greedy*, atau di dalam metoda pemecahan rantai penjumlahan terpendek ini sendiri lebih dikenal dengan nama metoda rantai penjumlahan terpendek.

Kata kunci: algoritma *greedy*, rantai penjumlahan terpendek.

1. PENDAHULUAN

Rantai penjumlahan merupakan masalah yang telah cukup lama ada. Rantai penjumlahan merupakan dasar dari operasi aritmatika. Rantai penjumlahan juga menjadi dasar dari operasi komputer, yaitu *increment* (penambahan) dan *decrement* (pengurangan).

Rantai penjumlahan didefinisikan sebagai urutan angka–angka a_0, a_1, a_2, a_3, ... yang memenuhi :

$$a_0 = 1$$

dan untuk setiap k > 0 :

$$a_k = a_i + a_j \text{ dimana i, j} < k$$

2. RANTAI PENJUMLAHAN

Semua bilangan dapat dicapai dengan menggunakan definisi rantai penjumlahan yang telah dijelaskan sebelumnya. Cara paling naif untuk mencari bilangan yang diinginkan dengan menggunakan rantai penjumlahan adalah dengan menjumlahkan 1 dengan bilangan yang telah ada sebelumnya hingga bilangan yang dimaksud dicapai.

Untuk memodelkan permasalahan ini, berikut diberikan rantai penjumlahan untuk mencari bilangan 4. urutan angka – angka yang dibentuk oleh rantai penjumlahan dengan cara naif adalah : 1,2,3,4 dengan rincian :

1 .....langkah 1
2 = 1 + 1.....langkah 2
3 = 2 + 1.....langkah 3
4 = 3 + 1.....langkah 4

Cara ini memang menjamin bahwa setiap bilangan pasti dapat ditemukan urutan angka – angkanya melalui rantai penjumlahan, tetapi cara ini juga sangat tidak mangkus apalagi untuk bilangan yang besar, misalnya 1000.

Telah banyak pendekatan yang dikembangkan untuk menemukan rantai penjumlahan untuk bilangan n dengan urutan angka – angka terpendek, tetapi belum ada algoritma yang dapat memberikan urutan angka – angka terpendek untuk sembarang bilangan n yang diberikan.

Meskipun begitu, beberapa metoda telah dikembangkan untuk menemukan urutan angka – angka yang relatif lebih pendek (mangkus) dibandingkan dengan cara naif yang disebutkan diatas.

Metoda yang akan dijelaskan di dalam makalah ini menggunakan algoritma *greedy*. Metoda ini dikenal juga dengan nama metoda rantai penjumlahan terpendek.

3. RANTAI PENJUMLAHAN TERPENDEK

Algoritma *greedy* diterapkan untuk menemukan rantai penjumlahan dengan urutan angka – angka terpendek. Ide dasar dari algoritma ini adalah dengan memanfaatkan urutan angka – angka yang telah terbentuk. Pada saat akan dibentuk sebuah angka baru, angka baru tersebut dibentuk dari angka dengan nilai terbesar yang mungkin, dengan harapan nilai tersebut dapat memberikan hasil yang optimum.

3.1 Algoritma *Greedy*

Algoritma *greedy* adalah algoritma yang memecahkan masalah langkah per langkah, dimana pada setiap langkah:

- mengambil pilihan terbaik yang dapat diperoleh pada saat itu (optimum lokal) tanpa memperhatikan konsekuensi ke depan
- berharap dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Pada setiap langkah di dalam algoritma *greedy*, kita baru menerima optimum lokal. Bila algoritma berakhir, kita berharap bahwa optimum lokal menjadi optimum global. Algoritma *greedy* mengasumsikan bahwa optimum lokal merupakan bagian dari optimum global. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah kembali pada langkah berikutnya.

Algoritma *greedy* memiliki elemen – elemen sebagai berikut :

1. Himpunan Kandidat, C : semua elemen pembentuk solusi.
2. Himpunan Solusi, S : semua solusi persoalan yang merupakan himpunan bagian dari himpunan kandidat.
3. Fungsi Seleksi : langkah – langkah untuk memilih kandidat yang menghasilkan solusi yang optimum.
4. Fungsi Kelayakan : fungsi yang akan mengembalikan nilai benar (*TRUE*) jika kandidat yang dipilih oleh fungsi seleksi bersama – sama dengan himpunan solusi tidak melanggar batasan yang ada.
5. Fungsi Objektif : Memaksimumkan / meminimumkan nilai solusi.

3.2 Penerapan Algoritma *Greedy*

Elemen – elemen pembentuk algoritma *greedy* dalam memecahkan permasalahan rantai penjumlahan terpendek ini, dipaparkan sebagai berikut :

1. Himpunan Kandidat, C
   Himpunan Kandidat dari permasalahan rantai penjumlahan ini adalah semua bilangan bulat yang dapat dibentuk berdasarkan penjumlahan 2 buah angka dari urutan angka – angka yang telah ada sebelumnya.
2. Himpunan Solusi, S
   Himpunan Solusi dari permasalahan rantai penjumlahan ini adalah bilangan hasil penjumlahan dengan nilai terbesar yang mungkin.
3. Fungsi (Predikat) Seleksi
   Fungsi Seleksi mengembalikan nilai bilangan yang memiliki nilai terbesar.
4. Fungsi (Predikat) Kelayakan
   Fungsi Kelayakan memeriksa apakah nilai yang diberikan bersama – sama dengan Himpunan Solusi sebelumnya telah melebihi bilangan yang hendak dicari.
5. Fungsi Obyektif
   Meminimumkan urutan angka – angka yang terbentuk.

Penjelasan algoritma *greedy* untuk menyelesaikan permasalahan rantai penjumlahan terpendek ini adalah sebagai berikut :

Berdasarkan urutan angka – angka yang telah ada sebelumnya a_0, a_1, a_2, a_3, ... dipilih angka terakhir (yaitu angka dengan nilai bilangan terbesar) a_n. Tambahkan a_n dengan dirinya sendiri :

$$a_{n+1} = a_n + a_n$$

Jika nilai a_{n+1} yang dihasilkan ternyata tidak diterima oleh fungsi kelayakan (nilai a_{n+1} melebihi nilai yang hendak dicari) maka diambil angka sebelum a_n untuk diuji apakah diterima oleh fungsi kelayakan atau tidak. Sehingga persamaan di atas menjadi :

$$a_{n+1} = a_n + a_{n-1}$$

Hal tersebut dilakukan hingga akhirnya ditemukan bilangan yang dimaksud.

Rancangan skema algoritma diatas diberikan dalam notasi algoritmik sebagai berikut :

```
  Kamus Global
  arrKandidat [] of integer {Himpunan
Kandidat yang akan dipilih}
  arrSolusi [] of integer {Himpunan
Solusi yang dihasilkan}

  Procedure RantaiPenjumlahan(input n
: integer)
  {prosedur untuk mencari urutan angka
– angka yang dapat dibentuk dari 1
hingga n dengan menggunakan algoritma
greedy}

  Kamus
  an : integer {bilangan yang akan
dicari berikutnya dalam usaha mencari
bilangan n}

  Algoritma
  an ← 1 {inisialisasi}
  while (an ≠ n) do
  {ulangi selama an belum sama dengan
nilai yang dicari}
     MakeKandidat() {menghasilkan
angka – angka yang mungkin dipilih
oleh fungsi seleksi}
     repeat
       an ← Seleksi() {fungsi Seleksi
untuk mengambil nilai bilangan
terbesar yang mungkin}
     until IsLayak(an)
     (fungsi isLayak memberikan true
jika nilai an dapat menjadi solusi dan
false jika tidak serta mengeluarkan
nilai tersebut dari himpunan
kandidat)
     AddSolusi (an)
     {Addsolusi menambahkan nilai an
ke dalam himpunan solusi yang ada}
  endwhile
  CetakSolusi(arrSolusi)
  {cetak solusi mencetak solusi yang
telah dihasilkan}
```

Gambar 1. Notasi algoritmik algoritma greedy untuk pencarian rantai penjumlahan terpendek

3.3 Contoh Penyelesaian Masalah

Sebagai gambaran dari Algoritma di atas, berikut diberikan beberapa contoh penyelesaian permasalahan rantai penjumlahan menggunakan algoritma tersebut.

Masalah : rantai penjumlahan untuk mencari angka 17

Langkah Penyelesaian :

- Pada saat awal, di dalam himpunan solusi hanya ada angka 1
- Dengan menggunakan himpunan solusi yang ada sebelumnya, dibentuk himpunan kandidat yang mungkin, yaitu 2 = 1 + 1.
- Fungsi Seleksi akan mengambil satu – satunya nilai yang ada di dalam himpunan kandidat saat ini, yaitu 2.
- Jika angka 2 ternyata telah melewati batas nilai yang dicari, maka angka 4 akan dikeluarkan dari himpunan kandidat dan langkah sebelumnya diulangi kembali.
- Oleh karena 2 masih belum melewati batas nilai angka yang dicari (yaitu 17) maka angka 2 dimasukkan ke dalam himpunan solusi. Saat ini himpunan Solusi berisi angka 1 dan 2
- Karena angka yang dicari (angka 17) belum ditemukan, maka langkah kedua diulangi kembali. Himpunan Kandidat dibentuk melalui himpunan solusi yang telah ada. Himpunan kandidat berisi : 2 = 1+1, 3 = 2 + 1,4 = 2 + 2.
- Fungsi Seleksi akan mengambil nilai terbesar yang ada pada himpunan kandidat, yaitu 4.
- Jika angka 4 ternyata telah melewati batas nilai yang dicari, maka angka 4 akan dikeluarkan dari himpunan kandidat dan langkah sebelumnya diulangi kembali.
- Oleh karena 4 masih belum melewati batas nilai yang dicari (yaitu 17) maka angka 4 dimasukkan ke dalam himpunan solusi. Saat ini himpunan Solusi berisi angka 1,2, 4.
- Langkah kedua diulangi kembali hingga akhirnya angka yang dicari ditemukan, yaitu angka 17.

Proses pencarian penyelesaian masalah rantai penjumlahan terpendek ini dapat dilihat dari tabel berikut :

Tabel 1 : Tabel proses pencarian solusi rantai penjumlahan berdasarkan algoritma *greedy*

Langkah	Keadaan				
	I	II	III	IV	V
0(INIT)	1	1	-	1	*TRUE*
1	2	2	1	1 , 2	*TRUE*
2	2 , 3 , 4	4	1 , 2	1 ,2 , 4	*TRUE*

3	2 , 3 , 4 , 5 , 6 , 8	8	1 , 2 , 4	1 , 2 , 4 , 8	*TRUE*
4	2 , 3 , 4 , 5 , 6 , 8 , 9 , 10 , 12 , 16	16	1 , 2 , 4 , 8	1 , 2 , 4 , 8 , 16	*TRUE*
5	2 , 3 ,4 ,5 , 6 , 8 , 9 , 1 0 , 12 , 16 , 17 , 19 , 20 , 21 , 22 , 24 , 25 , 26 , 28 , 32	32	1 , 2 , 4 , 8 , 16	1 , 2 , 4, 8, 16 , 32	*FALSE*
6	2 , 3 , 4 , 5 , 6 , 8 , 9 , 10 , 12 , 16 , 17 , 19 , 20 , 21 , 22 , 24 , 25 , 26 , 28	28	1 , 2 , 4 , 8 , 16	1 , 2 , 4, 8 , 16 , 28	*FALSE*
7	2 , 3 , 4 , 5 , 6 , 8 , 9 , 10 , 12 , 16 , 17 , 19 , 20 , 21 , 22 , 24 , 25 , 26	26	1 , 2 , 4 , 8 , 16	1 , 2 , 4 , 8 , 16 , 26	*FALSE*
8	2 , 3 , 4 , 5 , 6 , 8 , 9 , 10 , 12 , 16 , 17 , 19 , 20 , 21 , 22 , 24 , 25	25	1 , 2 , 4, 8 , 16	1 , 2 , 4 , 8 , 16 , 25	*FALSE*
9	2 , 3 , 4 , 5 , 6 , 8 , 9 , 10 , 12 , 16 , 17 , 19 , 20 , 21 , 22 , 24	24	1 , 2 , 4, 8 , 16	1 , 2 , 4 , 8 , 16 , 24	*FALSE*
10	2 , 3 , 4 , 5 , 6 , 8 , 9 , 10 , 12 , 16 , 17 , 19 , 20 , 21 , 22	22	1 ,2 ,4 , 8 , 16	1 , 2 , 4 , 8 , 16 , 22	*FALSE*
11	2 , 3 , 4 , 5 , 6 , 8 , 9 , 10 , 12 , 16 , 17 , 19 , 20 , 21	21	1 ,2 , 4 , 8 , 16	1 , 2 , 4 , 8 , 16 , 21	*FALSE*
12	2 , 3 , 4 , 5 , 6 , 8 , 9 , 10 , 12 , 16 , 17 , 19 , 20	20	1, 2 , 4 , 8 , 16	1 , 2 , 4 , 8 , 16 , 20	*FALSE*
13	2 , 3 , 4 , 5 , 6 , 8 , 9 , 10 , 12 , 16 , 17 , 19	19	1 , 2 , 4 , 8 , 16	1 , 2 , 4 , 8 , 16 , 19	*FALSE*
14	2 , 3 , 4 , 5 , 6 , 8 , 9 , 10 , 12 , 16 , 17	17	1 , 2 ,4 , 8 , 16 , 17	1 , 2 , 4 , 8 , 16 , 17	*TRUE*

Keterangan Tabel :
I : Himpunan Kandidat
II : Fungsi Seleksi
III : Himpunan Solusi Sebelum
IV : Himpunan Solusi Setelah
V : Fungsi Layak
INIT : Inisialisasi

3.4 Perbandingan dengan Algoritma *Bruteforce*

Untuk memberikan perbandingan penyelesaian masalah ini dengan algoritma lain, maka akan diberikan sebuah contoh yang akan diselesaikan dengan menggunakan algoritma *greedy* dan algoritma *bruteforce*.

Masalah : rantai penjumlahan untuk mencari angka 4

a. Dengan menggunakan algoritma *bruteforce*

Bruteforce adalah suatu cara penyelesaian masalah yang naif, yaitu dengan mencoba semua kemungkinan yang ada. Dalam permasalahan ini, kemungkinan yang ada adalah kombinasi dari bilangan mulai dari 1 hingga 3. Sehingga kemungkinan kombinasi yang akan dilakukan oleh algoritma brute force untuk menemukan solusi yang tepat adalah sebanyak 3! = 3 x 2 x 1 = 6.

Enumerasi dari langkah – langkah yang dilakukan oleh algoritma brute force ditunjukkan oleh tabel berikut :

Tabel 2 : Tabel proses pencarian solusi menggunakan algoritma *bruteforce*

Langkah	Himp. Solusi	Solusi()	Panjang
0(INIT)	1	*FALSE*	1
1	1 , 2	*FALSE*	2
2	1 , 2 , 3	*FALSE*	3
3	1 , 2 , 4	*TRUE*	3
4	1 , 2 , 3 , 4	*TRUE*	4
5	1 , 2 , 3 , 5	*FALSE*	4
6	1 , 2 , 3 , 6	*FALSE*	4

Keterangan Tabel :
INIT : Inisialisasi

b. Dengan menggunakan algoritma *greedy*

Untuk menyelesaikan masalah diatas, langkah – langkah yang diperlukan oleh algoritma *greedy* adalah sebagai berikut :

Tabel 3 : Tabel proses pencarian solusi menggunakan algoritma *greedy*

Langkah	Keadaan				
	I	II	III	IV	V
0(INIT)	1	1	-	1	*TRUE*
1	2	2	1	1 , 2	*TRUE*
2	2 , 3 , 4	4	1 , 2	1 , 2 , 4	*TRUE*

Keterangan Tabel :
I : Himpunan Kandidat
II : Fungsi Seleksi
III : Himpunan Solusi Sebelum
IV : Himpunan Solusi Setelah
V : Fungsi Layak
INIT : Inisialisasi

3.5 Analisa Hasil

Algoritma *greedy* terbukti memberikan hasil yang cukup memuaskan untuk masalah ini. Akan tetapi pada beberapa persoalan memungkinkan terjadinya perulangan pencarian akibat kegagalan himpunan solusi memenuhi persyaratan fungsi kelayakan. Contoh pada upabab 3.3 memperlihatkan salah satu kelemahan algoritma *greedy*

dalam menyelesaikan permasalahan ini (yaitu kurang mangkus untuk permasalahan kasus terburuk).

Meskipun begitu, algoritma *greedy* ini sendiri masih memberikan hasil yang lebih baik daripada algoritma *bruteforce*. Algoritma *bruteforce* akan mencoba segala kemungkinan yang ada berdasarkan himpunan solusi yang dapat dibentuknya, yaitu dari angka 1 hingga bilangan ke n yang hendak dicari. Dengan menggunakan algoritma *greedy* notasi Big-O nya adalah O(n) bandingkan dengan algoritma *bruteforce* yang dapat mencapai O(n!).

IV. KESIMPULAN

Meski algoritma *greedy* terbukti cukup memuaskan dan jauh lebih mangkus dibandingkan dengan algoritma *bruteforce*, banyak metoda – metoda lain yang diterapkan untuk menyelesaikan permasalahan ini. Salah satu algoritma lain yang digunakan untuk menyelesaikan permasalahan ini adalah algoritma runut balik. Dengan menggunakan algoritma runut balik, maka waktu pencarian yang diperlukan dapat dikurangi akan tetapi hasil yang diberikan adalah sama. Akan tetapi pada algoritma runut balik tersebut memori yang digunakan jauh lebih besar dibandingkan dengan memori yang digunakan oleh algoritma *greedy*, sehingga untuk kasus – kasus dimana memori menjadi kendala, algoritma *greedy* lebih dipertimbangkan daripada algoritma runut balik untuk memecahkan masalah ini. Pertimbangan lain adalah disebabkan waktu proses yang diberikan oleh keduanya tidak berbeda terlalu jauh, meski algoritma runut balik masih relatif lebih cepat.

REFERENSI

[1] Rinaldi Munir, "Diktat Kuliah Strategi Algoritmik", Program Studi Teknik Informatika ITB, 2006.

[2] http://en.wikipedia.org/wiki/Addition_chain, diakses pada tanggal 18 Mei 2008 pukul 19:22.

[3] http://wwwhomes.uni-bielefeld.de/achim/addition_chain.html, diakses pada tanggal 18 Mei 2008 pukul 19:47.

[4] http://mathworld.wolfram.com/AdditionChain.html, diakses pada tanggal 18 Mei 2008 pukul 20:20.

[5] http://www-cs-staff.stanford.edu/~knuth/programs/strongchain.w, diakses pada tanggal 19 Mei 2008 pukul 19:10.