Pengembangan Algoritma Greedy untuk Game Treasure Hunter

Michael Alexander Wangsa 13512046
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
michaelaw320@gmail.com

***Abstract*—Algoritma greedy**

***Index Terms*—algoritma, greedy, kendala, optimisasi.**

I. PENDAHULUAN

Algoritma adalah kumpulan prosedur atau perintah atau urutan pengerjaan dalam menyelesaikan suatu masalah.

Ada banyak permasalahan yang dapat diselesaikan oleh algoritma-algoritma.

Contoh dari permasalahan yang membutuhkan algoritma adalah bagaimana cara memilih jalan tersingkat dari satu titik ke titik lain yang di modelkan dalam bentuk graf.

Ada dua cara menyelesaikan permasalahan diatas yaitu dengan cara menghitung satu per satu kemungkinan solusi lalu memilih solusi yang paling singkat. Akan tetapi cara seperti ini tidak efisien seiring dengan meningkatnya jumlah jarak yang harus dihitung meski menjamin hasil yang optimal.

Oleh karena itu, dikembangkanlah cara cara atau metode lain yang memungkinkan mendapatkan hasil yang cukup optimal atau bahkan optimal tetapi memakan waktu sesedikit mungkin, dalam persoalan diatas, algoritma greedy bisa dipakai untuk menyelesaikan masalah dengan cepat dan optimal.

Algoritma greedy adalah algoritma yang sederhana dan berusaha mengambil sebanyak mungkin solusi tanpa mementingkan konsekuensi berikutnya. Algoritma greedy biasa digunakan sehari hari dalam kasus memilih jalan terpendek dari kota A menuju kota B, memilih jurusan perguran tinggi, maupun dalam bermain kartu remi.

II. DASAR TEORI

Algoritma greedy membentuk solusi langkah per langkah. Setiap langkah yang dilalui harus diambil satu keputusan terbaik pada saat itu dianggap paling baik. Keputusan yang telah diambil sebelumnya tidak bisa diubah pada langkah berikutnya.

Pendekatan yang digunakan dalam algoritma greedy adalah memilih pilihan yang sepertinya memberikan hasil terbaik yaitu dengan membuat pilihan optimum lokal pada setiap langkah dengan harapan mendapatkan solusi yang merupakan optimum global dari serangkaian optimum optimum lokal.

Kesimpulan dari pendekatan pendekatan diatas adalah algoritma greedy adalah algoritma yang memecahkan masalah dalam langkah per langkah dengan setiap langkahnya :

- Mengambil pilihan terbaik yang dapat dipilih saat langkah n tanpa menimbang langkah selanjutnya.
- Berharap dengan memilih optimum lokal pada setiap langkah n menjadi optimum global. Algoritma greedy mengasumsikan seluruh rangkaian optimum lokal merupakan bagian dari optimum global.

Persoalan optimasi dalam algoritma greedy disusun oleh elemen elemen sebagai berikut:

1. Himpunan Kandidat
2. Himpunan Solusi
3. Fungsi seleksi
4. Fungsi kelayakan
5. Fungsi obyektif

Himpunan Kandidat adalah himpunan yang berisi elemen pembentuk solusi. Pada setiap langkah diambil satu buat kandidat dari himpunan ini.

Himpunan Solusi adalah himpunan yang berisi kandidat-kandidat terpilih sebagai solusi persoalan.

Fungsi seleksi adalah fungsi yang bertugas memilih kandidat yang paling memungkinkan mencapai solusi optimal. Kandidat yang sudah terpilih tidak akan diikutsertakan lagi dalam penyeleksian pada tahap selanjutnya.

Fungsi kelayakan memeriksa apakah kandidat yang dipilih dapat memberikan solusi yang layak. Kandidat diuji oleh fungsi kelayakan agar tidak melebihi kendala yang ditetapkan, jika melanggar, kandidat ditolak.

Fungsi obyektif adalah fungsi yang memaksimumkan atau meminimumkan nilai solusi, misal panjang lintasan, keuntungan, biaya, dll.

Algoritma greedy harus memenuhi kriteria kriteria di atas untuk mendapatkan solusi optimal dari suatu masalah.

Adapun skema umum cara kerja algoritma greedy

adalah sebagai berikut:

- Inisialisasi S dengan kosong
- Pilih kandidat dengan fungsi seleksi dari C
- Kurangi C dengan kandidat terpilih
- Periksa kelayakan kandidat yang dipilih dengan himpunan solusi apakah membentuk solusi yang tidak melanggar fungsi kelayakan, jika melanggar, maka buang kandidat pilih kandidat lain, jika tidak melanggar, masukkan kandidat ke dalam S.
- Periksa apakah himpunan solusi memberikan solusi yang lengkap, jika lengkap, maka berhenti dan solusi didapatkan, bila tidak, ulang dari langkah memilih kandidat dengan fungsi seleksi dari C

Algoritma greedy tidak selalu menghasilkan solusi yang seratus persen optimal, hal ini dikarenakan algoritma greedy tidak beroperasi secara menyeluruh terhadap semua alternative solusi seperti teknik *exhaustive search* yang memakan banyak waktu.

Faktor lain yang menyebabkan algoritma greedy tidak selalu mendapatkan solusi optimal yaitu pemilihan fungsi seleksi. Dalam satu persoalan mungkin terdapat beberapa fungsi seleksi yang dapat memberikan hasil yang berbeda untuk persoalan yang sama. Fungsi seleksi yang paling tepat lah yang harus dipilih agar solusi benar benar optimum seperti yang diinginkan.

Karena kedua factor diatas, algoritma greedy bisa dikatakan hanya memberikan hampiran optimum. Jika suatu algoritma greedy dikatakan menghasilkan solusi optimum, harus terdapat bukti matematis yang menunjukkan bahwa benar algoritma tersebut menghasilkan solusi optimal.

III. GREEDY TREASURE HUNTER

Dalam permainan *Greedy Treasure Hunter*, pemain akan mengumpulkan koin dari peti harta karun. Area permainan digambarkan sebagai graf dengan jarak antar simpul yang terdefinisi. Simpul dapat merepresentasikan posisi pemain, posisi semua peti, atau posisi kosong. Waktu yang dibutuhkan untuk berpindah dari satu simpul ke simpul lainnya sesuai dengan jarak yang dibutuhkan, misalnya jarak pemain ke suatu peti 5, artinya membutuhkan 5 satuan waktu.

Terdapat 3 warna yang berbeda untuk peti harta karun. Peti merah berisi 5 koin, peti kuning berisi 3 koin, dan peti hijau berisi 1 koin. Untuk mendapatkan koin dari suatu peti, pemain harus menggunakan tipe peralatan yang tepat untuk membuka peti karena peti akan meledak jika tidak terbuka dalam waktu tertentu. Semakin banyak koin di dalam peti, semakin cepat peti tersebut akan meledak. Peti merah akan meledak dalam 1 satuan waktu, peti kuning akan meledak dalam 3 satuan waktu, dan peti hijau akan meledak dalam 4 satuan waktu.

Di awal permainan, pemain mendapat modal uang untuk membeli sejumlah peralatan yang tersedia. Peralatan dibagi menjadi 3 tipe yaitu normal, 2x, 4x. Peralatan tipe normal akan membuka peti selama 4 satuan waktu (hanya bisa untuk peti hijau), sedangkan peralatan tipe 2x atau 4x akan membuka peti lebih cepat 2x (2 satuan waktu; untuk peti kuning atau hijau) atau 4x (1 satuan waktu; untuk semua peti). Peralatan yang sudah dipakai tidak dapat digunakan lagi untuk membuka peti yang lain, sehingga pemain perlu memilih jumlah peralatan yang akan dibeli untuk setiap tipe peralatan. Harga peralatan normal adalah satu koin, sedangkan peralatan 2x adalah dua koin, dan harga peralatan 4x adalah tiga koin.

Dalam permainan, didefinisikan area permainan, modal awal pemain, target koin yang harus dikumpulkan oleh pemain, jumlah peti untuk setiap tipenya. Posisi pemain dan posisi semua peti dibangkitkan secara random. Skor pada permainan ini adalah sisa modal awal ditambah jumlah koin yang didapatkan untuk durasi waktu tertentu. Pemain harus berusaha menggunakan modal sesedikit mungkin tetapi memperoleh koin sebanyak-banyaknya.

Akan disimulasikan permainan dengan menggunakan algoritma greedy. Komputer akan bermain game ini sendirian menggunakan algoritma greedy sebagai metode pemilihan langkah.

IV. ALGORITMA MAGI

Penulis untuk memenuhi persyaratan permainan diatas membuat rangkaian algoritma yang berfungsi untuk mencari jalan terbaik dengan memilih rasio nilai peti / jarak sebagai fungsi seleksi, lalu menguji apakah masih tersedia bahan bakar untuk menuju ke tujuan sebagai fungsi kelayakan serta tambahan fungsi kelayakan berupa pengecekan apakah peti akan meledak jika dibuka / tersedia alat pembuka peti atau tidak.

Berikut header dari fungsi-fungsi dalam Bahasa C++

```
int MAGI(Graf G, Player P, int TimeLeft);

int* Melchior(Graf G, Player P,int*
ArrayJarak);
/* Mengusulkan kandidat simpul yang
dikunjungi berikutnya */

bool Balthazar(int* ArrayJarak, int Simpul,
int TimeLeft);
/* Menolak atau menerima kandidat tujuan
simpul berdasarkan waktu tempuh*/

bool Casper(Graf G, Player P, int TimeLeft,
int* ArrayJarak, int Simpul);
/* Menolak atau menerima kandidat tujuan
berdasarkan ketersediaan alat pembuka peti
dan mengecek apakah peti meledak jika
dicoba dibuka*/
```

Fungsi greedy secara keseluruhan adalah fungsi MAGI yang mengembalikan sebuah nilai integer yang merupakan node yang akan dituju. Fungsi MAGI menerima input berupa sebuah Graf G, data keadaan Player sekarang untuk mengetahui posisi player saat ini serta kepemilikan alat pembuka peti player, serta sebuah integer TimeLeft yang merupakan sisa waktu/bahan bakar yang tersisa.

Berikut ini adalah source code untuk algoritma MAGI

```
int MAGI(Graf G, Player P, int TimeLeft){
   int n = G.GetJumlahSimpul();
   int* ArrayJarak;
   ArrayJarak = new int[n];
   for (int i=0;i<n;i++) {
      ArrayJarak[i] =
G.GetMatrixJarakSimpul()[P.GetCurrentPos()][i];
   }

   int* Kandidat;
   Kandidat = Melchior(G,P, ArrayJarak);

   int i = 0;
   bool stop= false;
         bool lock = false;
   int MAGIDECISION = Kandidat[0];
   int min = ArrayJarak[Kandidat[0]];
   while ((i < n) && !stop) {
      bool M2,M3;
      M2 = Balthazar(ArrayJarak, Kandidat[i],
TimeLeft);
      if (M2) {
         if (ArrayJarak[Kandidat[i]] < min &&
ArrayJarak[Kandidat[i]] != -99 && !lock &&
P.GetCurrentPos() != Kandidat[i]) {
            MAGIDECISION = Kandidat[i];
            min = ArrayJarak[Kandidat[i]];
                                    lock = true;
         }
         M3 = Casper(G, P, TimeLeft, ArrayJarak,
Kandidat[i]);
         if (M3) {
            MAGIDECISION = Kandidat[i];
            stop = true;
         }
                           else {
                                    i++;
                           }
      } else {
         i++;
      }
   }
   /* Cleanup code */
   delete[] ArrayJarak;
   return MAGIDECISION;
}
```

Skema kerja umum MAGI adalah :

- Melakukan copy sebuah array dari matrix ketetanggaan Graf G dari posisi player saat ini menghasilkan ArrayJarak, dimana ArrayJarak[j] = Matrix(posisi player, j).
- Memanggil fungsi Melchior yang bertugas membangun himpunan kandidat berupa rasio nilai yang akan didapatkan / jarak tempuhnya yang terurut menurun dari rasio terbesar (paling menguntungkan) sampai titik simpul yang tidak dapet dilalui
- Memanggil fungsi Balthazar untuk mengevaluasi apakah kandidat masih dapat dicapai dengan waktu yang tersisa
- Jika Balthazar mengembalikan nilai true, dipanggil Casper untuk mengevaluasi ketersediaan alat serta waktu apakah memadai untuk membuka peti

Skema umum MAGI memiliki kompleksitas $O(n^2)$ dikarenakan pengurutan menggunakan bubble sort.

Jika diinginkan lintasan utuh dari awal sampai akhir maka dibutuhkan kompleksitas $O(n^3)$ karena MAGI diulang lagi sebanyak n kali.

Kasus terbaik dari MAGI adalah ketika kandidat dengan rasio terbesar dipilih dengan waktu tersedia serta alat dan waktu tersedia, kasus terburuk adalah ketika semua pilihan ditolak oleh Balthazar yaitu ketika waktu habis / tidak cukup.

```
int* Melchior(Graf G, Player P, int* ArrayJarak) {
   int n = G.GetJumlahSimpul();
   double* rasio;
   int* posisi;
   rasio = new double[n];
   posisi = new int[n];
   for (int i = 0; i < n; i++) {
      posisi[i] = i;
      if(ArrayJarak[i] == 0) {
         if(G.GetPetiPosKe(i) == 0) rasio[i] = -99;
                                 else  {
                                          if
(P.GetNDrill() == 0 && P.GetNRailgun() == 0 &&
P.GetNLaser() == 0) rasio[i] = -99;
                                          else rasio[i] =
999; //prioritas
                                 }
      }
      else if (ArrayJarak[i] == -99) rasio[i] = -99;
      else {
         int x;
         if (G.GetPetiPosKe(i) == 0) x = 0;
         else if(G.GetPetiPosKe(i) == MERAH){ x
= SOLAR_RED;}
         else if (G.GetPetiPosKe(i) == KUNING){ x
= SOLAR_YELLOW;}
         else if (G.GetPetiPosKe(i)== HIJAU){ x =
SOLAR_GREEN;}
            rasio[i] = (double) x/ArrayJarak[i];
      }
   }
   int i,j;
   double temp;
   int tempos;
   for(i=0;i<n;i++) {
      for ( j = 0;j< n-1;j++) {
         if((rasio[j]) < rasio[j+1]) {
            temp = rasio[j+1];
            rasio[j+1] = rasio[j];
            rasio[j] = temp;
            tempos = posisi[j+1];
            posisi[j+1] = posisi[j];
            posisi[j] = tempos;
         }
      }
   }
   /* Cleanup */

   delete[] rasio;
   return posisi;
}
```

Skema kerja Melchior secara umum:

- Menghitung keuntungan yang didapat didasarkan dari nilai peti yang akan didapatkan dibagi dengan jarak yang dari simpul player saat ini ke simpul yang dapat dituju

- Perhitungan diulang sebanyak n buah simpul yang dapat dijangkau dari player dan dimasukkan ke dalam larik.
- Larik kemudian diurutkan dengan bubble sorting untuk mendapatkan rasio terbesar
- Pada saat yang bersamaan dilakukan pengurutan nomor simpul mengikuti rasio terbesar
- Hasil kembalian fungsi berupa nomor urut simpul yang sudah terurut menurun berdasarkan rasio

Oleh karena pengurutan menggunakan bubble sort, kompleksitas algoritma ini adalah $O(n^2)$.

```
bool Balthazar (int* ArrayJarak, int Simpul, int
TimeLeft) {
   int a;
   if(ArrayJarak[Simpul] == -99) return false;
   else if (ArrayJarak[Simpul] == 0) return false;
   else {
      a = TimeLeft - ArrayJarak[Simpul];
      return (a >= 0);
   }
}
```

Skema kerja fungsi Balthazar secara umum:

- Membuat keputusan apakah simpul yang di rekomendasikan oleh Melchior memenuhi persyaratan kendala waktu, apabila memenuhi akan mengembalikan true jika tidak mengembalikan false

Jika semua kandidat ditolak oleh Balthazar berarti proses dihentikan dan solusi terakhir merupakan lintasan terakhir yang dapat dilalui.

```
bool Casper(Graf G, Player P, int TimeLeft, int*
ArrayJarak, int Simpul) {
   int val;
   if (G.IsKosong(G.GetPetiPosKe(Simpul)) &&
(Simpul != P.GetCurrentPos())) return true;
   else if (G.IsMerah(G.GetPetiPosKe(Simpul))
&& (P.GetNRailgun() >= 1)) {
      val = TIME_RED;
   }
   else if (G.IsKuning(G.GetPetiPosKe(Simpul))
&& (P.GetNRailgun() >= 1)){
      val = TIME_RED;
   }
   else if (G.IsKuning(G.GetPetiPosKe(Simpul))
&& (P.GetNLaser() >= 1)){
      val = TIME_YELLOW;
   }
   else if (G.IsHijau(G.GetPetiPosKe(Simpul)) &&
(P.GetNRailgun() >= 1)){
      val = TIME_RED;
   }
   else if (G.IsHijau(G.GetPetiPosKe(Simpul)) &&
(P.GetNLaser() >= 1)){
      val = TIME_YELLOW;
   }
   else if (G.IsHijau(G.GetPetiPosKe(Simpul)) &&
(P.GetNDrill() >= 1)){
      val = TIME_GREEN;
   }
   else return false;

   int a;
   a = TimeLeft - (ArrayJarak[Simpul]+val);
   if (a < 0) {
      return false;
   }


   return true;
}
```

Skema kerja fungsi Casper secara umum:

- Menghitung apakah ketika sampai di tujuan peti dapat dibuka atau meledak karena alat tidak tersedia
- Menghitung waktu sisa apakah memungkinkan waktu sisa dipakai untuk melakukan perjalanan sekaligus membuka peti.

Keputusan Casper tidak begitu diperhatikan oleh algoritma MAGI karena kurang krusial, Casper biasanya berfungsi secara baik apabila tersisa pilihan yang memiliki kendala waktu yang cukup ketat.

V. CONCLUSION

Algoritma MAGI yang dikembangkan oleh penulis untuk memenuhi kasus Greedy Treasure Hunter dapat dikategorikan lebih optimal dibandingkan algoritma yang hanya menghitung berdasarkan jarak lintasan terpendek dan memiliki kompleksitas $O(n^3)$.

VII. ACKNOWLEDGMENT

Penulis mengucapkan syukur terima kasih kepada Tuhan yang Maha Esa dan anugrah-Nya yang memungkinkan penulis menulis makalah ini.

Penulis juga berterima kasih kepada Dr. Ir. Rinaldi Munir, M.T. dan Dr. Masayu Leylia Khodra, S.T, M.T. atas pelajaran yang telah diberikan selama satu semester lamanya penulis dapat diajar oleh beliau.

Penulis berterimakasih kepada orang tua penulis yang medukung penulis dalam pembuatan makalah ini.

Penulis juga berterima kasih kepada teman-teman penulis yang mendukung serta memberikan ide dalam membuat makalah ini.

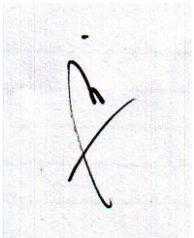
REFERENCES

Munir, Rinaldi (2009) "Diktat Kuliah Strategi Algoritma"

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, 19 Mei 2014



Michael Alexander Wangsa 13512046