Kelompok : 1. M.Ari Gunawan 18.11.2075
2. Aji Saputro 18.11.2042
3. Wahyu Ulta Pratasa Dewa 18.11.2055
Kleas : IF-04
Tugas Ai : Heuristic Search

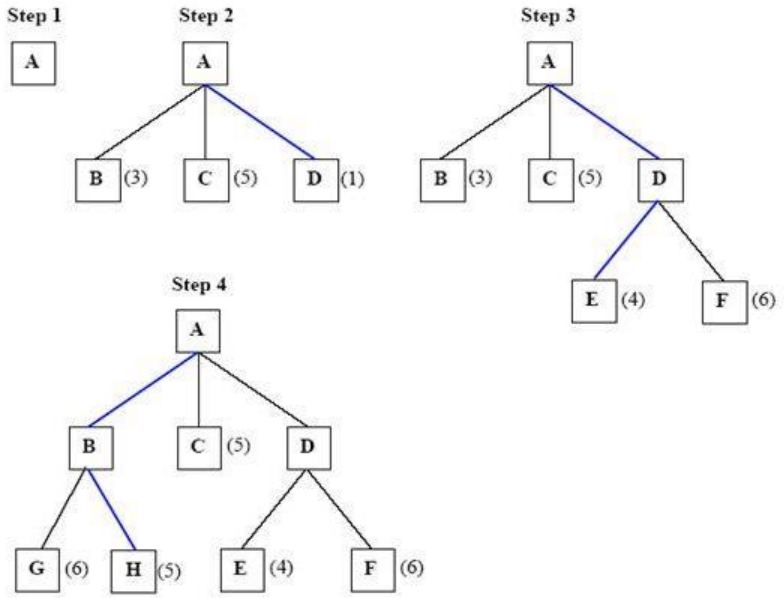
Best-First Search

Pengertian Best-first Search

Best-First Search merupakan sebuah metode yang membangkitkan simpul dari simpul sebelumnya. Best-first search memilih simpul baru yang memiliki biaya terkecil diantara semua leaf nodes (simpul-simpul pada level terdalam) yang pernah dibangkitkan. Penentuan simpul terbaik dilakukan dengan menggunakan sebuah fungsi yang disebut fungsi evaluasi f(n). fungsi evaluasi best-first search dapat berupa biaya perkiraan dari suatu simpul menuju ke goal atau gabungan antara biaya sebenarnya dan biaya perkiraan tersebut.

Algoritma best-first search

Pertama kali, dibangkitkan node A. Kemudian semua suksesor A dibangkitan, dan dicari harga paling minimal. Pada langkah 2, node D terpilih karena harganya paling rendah, yakni 1. Langkah 3, semua suksesor D dibangkitkan, kemudian harganya akan dibandingkan dengan harga node B dan C. Ternyata harga node B paling kecil dibandingkan harga node C, E, dan F. Sehingga B terpilih dan selanjutnya akan dibangkitkan semua suksesor B. Demikian seterusnya sampai ditemukan node tujuan. Ilustrasi algoritma best-first search dapat dilihat pada gambar 3.4 dibawah ini.



Untuk mengimplementasikan algoritma pencarian ini, diperlukan dua buah senarai, yaitu: OPEN untuk mengelola node-node yang pernah dibangkitkan tetapi belum dievaluasi dan CLOSE untuk mengelola node-node yang pernah dibangkitkan dan sudah dievaluasi. Algoritma selengkapnya adalah sebagai berikut.

1. OPEN berisi initial state dan CLOSED masih kosong.
2. Ulangi sampai goal ditemukan atau sampai tidak ada di dalam OPEN.
   a. Ambil simpul terbaik yang ada di OPEN.
   b. Jika simpul tersebut sama dengan goal, maka sukses
   c. Jika tidak, masukkan simpul tersebut ke dalam CLOSED
   d. Bangkitkan semua aksesor dari simpul tersebut
   e. Untuk setiap suksesor kerjakan:

i. Jika suksesor tersebut belum pernah dibangkitkan, evaluasi suksesor tersebut, tambahkan ke OPEN, dan catat parent.

ii. Jika suksesor tersebut sudah pernah dibangkitkan, ubah parent-nyajika jalur melalui parent ini lebih baik daripada jalur melalui parent yang sebelumnya. Selanjutnya perbarui biaya untuk suksesor tersebut dn nodes lain yang berada di level bawahnya.

Algoritma A* (A-Star)

Konsep Algoritma A* (A-Star)

(A Star) Dalam sains komputer, A* (dibaca “A star”) adalah algoritma komputer yang digunakan secara luas dalam mencari jalur (path finding) dan grafik melintang (graph traversal), proses plotting sebuah jalur melintang secara efisian antara titik- titik, disebut node. Terkenal karena penampilan dan akurasinya, algoritma ini diperluas untuk berbagai bidang. A* mencapai penampilan yang lebih baik dengan menggunakan heuristik. A* menggunakan Best First Search (BFS) dan menemukan jalur dengan biaya terkecil (least-cost path) dari node awal (initial node) yang diberikan ke node tujuan (goal node). Algoritma ini menggunakan fungsi heuristik jarak ditambah biaya (biasa dinotasikan dengan f(x)) untuk menentukan urutan di mana search-nya melalui node-node yang ada di pohon (tree).

Notasi yang dipakai oleh Algoritma A* adalah sebagai berikut:

$f(n) = g(n) + h(n)$

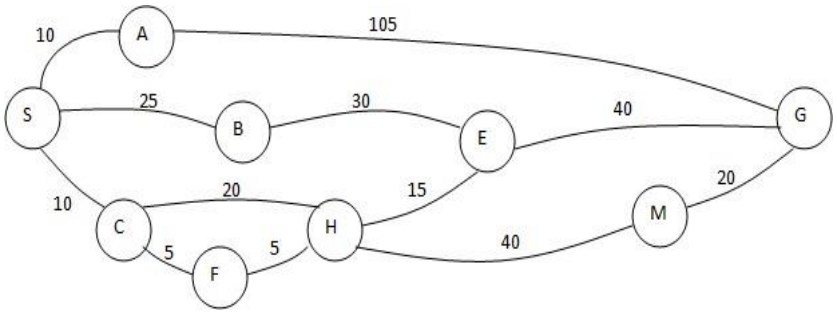
f(n) = biaya estimasi terendah

g(n) = biaya dari node awal ke node n

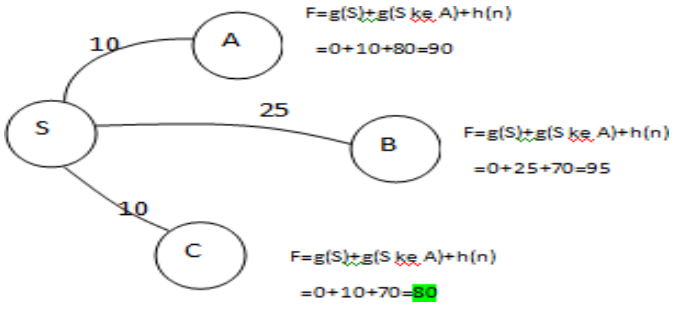
h(n) = perkiraan biaya dari node n ke node akhir.

Dalam penerapannya, Algoritma A* memiliki beberapa terminologi dasar diantaranyastarting point, simpul (nodes), A, open list, closed list, harga (cost), halangan (unwalkable).

Simulasi Algoritma A* (A-Star)



n	S	A	B	C	E	F	G	H	M
h(n)	80	80	70	70	75	78	0	70	70



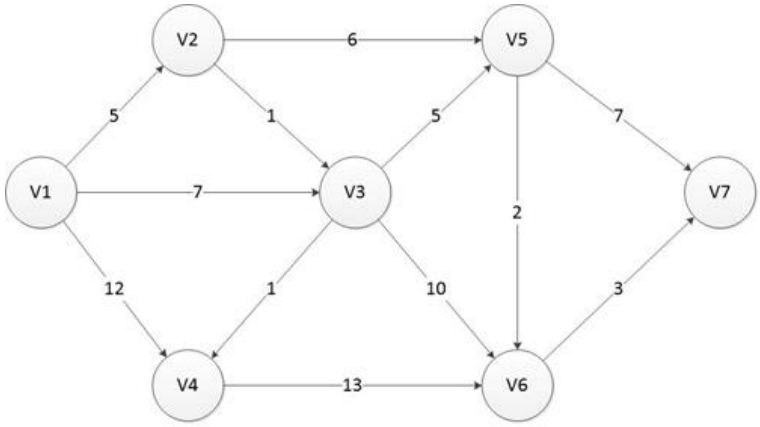
Notasi Algoritma A* (A-Star)

Pseudo Code A*

```
function A* (masalah) returns solusi
        OPEN <- S
        CLOSED <- array kosong
loop sampai ditemukan atau sampai tidak ada
        if OPEN = kosong then
          Gagal
else
        BestNode simpul yang ada di OPEN
        dengan f minimal
        Pindahkan simpul terbaiktersebut dari
        OPEN ke CLOSED
        if BestNode goal then
          Sukses
  else
        Bangkitkan semua suksesor BestNode
        tapi jangan buat pointer
        Untuk setiap suksesor kerjakan:
        Hitung g(suksesor) g(BestNode) +
        actual cost(dari BestNode ke suksesor)
        {Periksa suksesor}
        if suksesor ada di OPEN then (sudah
pernah dibangkitkan tapi belum diproses)
          OLD simpul di OPEN yang sama
          dengan suksesor tersebut
          Tambahkan OLD sebagai suksesor
          BestNode
          Buat pointer dari OLD ke BestNode
          Bandingkan nilai g(OLD) dengan
          g(suksesor)
          if g(OLD) lebih baik then
                Ubah parent OLD ke
                BestNode
                Ubah nilai g dan f yang ada
                pada OLD
          end
  else
          if suksesor ada di CLOSED then
(sudah pernah dibangkitkan dan sudah
diproses)
                OLD simpul di CLOSED yang
                sama dengan suksesor
                tersebut
                Tambahkan OLD sebagai
                suksesor BestNode
                Bandingkan nilai g(OLD)
                dengan g(suksesor)
                if g(OLD) lebih baik then
                  Ubah parent OLD ke
                  BestNode
                  Ubah nilai g dan f yang
                  ada pada OLD
                  Propagasi untuk semua
                  suksesor OLD dengan
                  penelusuran DFS dengan
                  aturan:
                  loop sampai simpul sukseso
                  tidak ada di OPEN atau simp
                  tidak punya suksesor
                    if suksesor ada di OPEN
then
                          Propagasi
diteruskan
                    else
                          if nilai g via
suksesor lebih baik then
                                Propagasi
diteruskan
                          else
                                Propagasi
dihentikan
                    end
                end
   end
  else {suksesor tidak ada di OPEN maupun
CLOSED}
        Masukkan suksesor ke OPEN
        Tambahkan suksesor tersebut sebagai
        suksesornya BestNode
        Hitung f: g(suksesor) + h(suksesor)
    end
end
end
end
```

Algoritma Djikstra

Algoritme Dijkstra, (sesuai penemunya Edsger Dijkstra), adalah sebuah algoritma yang dipakai dalam memecahkan permasalahan jarak terpendek (shortest path problem) untuk sebuah graf berarah (directed graph).



Permasalahan rute terpendek dari sebuah titik ke akhir titik lain adalah sebuah masalah klasik optimasi yang banyak digunakan untuk menguji sebuah algoritma yang diusulkan. Permasalahan rute terpendek dianggap cukup baik untuk mewakili masalah optimisasi, karena permasalahannya mudah dimengerti (hanya menjumlahkan seluruh edge yang dilalui) namun memiliki banyak pilihan solusi.
Menurut Andrew Goldberg peneliti Microsoft Research Silicon Valley, mengatakan ada banyak alasan mengapa peneliti terus mempelajari masalah pencarian jalan terpendek. “Jalan terpendek adalah masalah optimasi yang relevan untuk berbagai macam aplikasi, seperti jaringan routing, game, desain sirkuit, dan pemetaan”.

Diskripsi matematis untuk grafik dapat diwakili G = {V. E}, yang berarti sebuah grafik (G) didefenisikan oleh satu set simpul (Vertex = V) dan koleksi Edge (E). Algoritma Dijkstra bekerja dengan membuat jalur ke satu simpul optimal pada setiap langkah. Jadi pada langkah ke n, setidaknya ada n node yang sudah kita tahu jalur terpendek. Langkah-langkah algoritma Dijkstra dapat dilakukan dengan langkah-langkah berikut:

1. Tentukan titik mana yang akan menjadi node awal, lalu beri bobot jarak pada node pertama ke node terdekat satu per satu, Dijkstra akan melakukan pengembangan pencarian dari satu titik ke titik lain dan ke titik selanjutnya tahap demi tahap.

2. Beri nilai bobot (jarak) untuk setiap titik ke titik lainnya, lalu set nilai 0 pada node awal dan nilai tak hingga terhadap node lain (belum terisi) 2.

3. Set semua node yang belum dilalui dan set node awal sebagai “Node keberangkatan”.

4. Dari node keberangkatan, pertimbangkan node tetangga yang belum dilalui dan hitung jaraknya dari titik keberangkatan. Jika jarak ini lebih kecil dari jarak sebelumnya (yang telah terekam sebelumnya) hapus data lama, simpan ulang data jarak dengan jarak yang baru.

5. Saat kita selesai mempertimbangkan setiap jarak terhadap node tetangga, tandai node yang telah dilalui sebagai “Node dilewati”. Node yang dilewati tidak akan pernah di cek kembali, jarak yang disimpan adalah jarak terakhir dan yang paling minimal bobotnya.

6. Set “Node belum dilewati” dengan jarak terkecil (dari node keberangkatan) sebagai “Node Keberangkatan” selanjutnya dan ulangi langkah e.