Pencarian Tanpa Informasi

2.4 Pencarian

Pencarian di dalam sebuah ruang pencarian memerlukan hal berikut:

- Himpunan keadaan
- Operator dan biaya
- Keadaan awal
- Tes untuk menentukan tujuan tercapai atau tidak

2.4.1 Algoritma Pencarian Umum

```
Ambil sebuah List L, sebagai keadaan awal (L = fringe)
Loop
   if L kosong return failure
   Node ← select (L)
   if Node adalah sebuah tujuan
   then return Node
  (path dari keadaan awal ke Node)
  else buka semua suksesor dari Node, dan satukan
  keadaan ke dalam L
End Loop
```

Setiap keadaan yang baru dibuka harus selalu dicatat, dan keadaan ini disebut dengan simpul (Node). Data struktur yang dipakai oleh sebuah simpul akan dipakai untuk menelusuri keadaan baru, dan juga keadaan sebelumnya, sesuai dengan langkah yang diambil melalui operator. Setiap algoritma pencarian akan menggunakan sebuah daftar (List) dari semua simpul, disebut sebagai fringe. Fringe ini dipakai sebagai alat untuk menunjukkan simpul yang telah dibuka dan siap untuk dieksplorasi.

Pada awalnya, fringe akan berisi sebuah simpul tunggal, yang merupakan keadaan awal, dalam keadaan OPEN, siap diekslorasi. Algoritma akan mengambil simpul pertama dari dalam fringe untuk dibuka. Jika simpul merupakan tujuan, maka path yang telah terdata di dalam simpul merupakan hasil pencarian. Jalur untuk mencapai simpul tujuan dapat ditemukan dengan menelusuri dari himpunan keadaan yang dilalui sebelumnya (Parent).

Suksesor dari keadaan saat ini, akan dimasukkan ke dalam fringe, yang kemudian dapat dibuka untuk mencari simpul (keadaan) lainnya.

2.4.2 Fundamental Algoritma Pencarian

Mengacu pada sebuah algoritma pencari, sebuah pohon pencarian akan berisi simpul yang telah dibuka dan telah dieksplorasi. Pohon pencarian mungkin saja sangat luas, hal ini dapat terjadi jika terdapat pengulangan (loop) pada keadaan. Bagaimana pengulangan dapat dihindari?

Mengacu kembali pada sebuah algoritma pencarian, apakah yang dihasilkan, sebuah jalur atau simpul? Jawabannya tergantung pada jenis permasalahan. Untuk problem seperti N-queens, yang diperlukan adalah simpul keadaan akhir. Untuk permasalahan sejenis 8-puzzle, yang menarik untuk dicari adalah jalurnya.

Pada pencarian algoritma umum, terlihat harus dipilih sebuah simpul untuk dibuka. Simpul mana yang harus dibuka? Kapan harus ditempatkan simpul baru pada fringe? Tergantung pada problem yang hendak dipecahkan, akan terdapat beberapa kasus. Graf pencarian mungkin saja diberikan bobot (nilai) ataupun tidak. Pada beberapa kasus, diperlukan informasi mengenai kualitas dari keadaan yang dilalui selama pencarian, dan hak ini dapat dieksploitasi melalui algoritma pencarian. Bergantung pada persoalan pula, tujuan yang hendak dicapai dapat dinilai melalui biaya jalur pencarian yang terbentuk,misalnya yang perlu dicari adalah jalur dengan biaya terendah atau jarak terpendek, dsb.

2.4.3 Evaluasi Strategi Pencarian

Karakteristik algoritma pencarian akan dapat dinilai melalui beberapa faktor, yaitu:

1. Kelengkapan: apakah dapat diberikan garansi bahwa solusi terdapat dalam ruang pencarian?
2. Optimasi: apakah solusi yang akan didapatkan hanya memerlukan biaya minimal?
3. Bagaimana dengan penghitungan biaya pencarian, dikaitkan dengan waktu dan ruang pencarian?
    a. Kompleksitas waktu: Waktu yang diperlukan (jumlah simpul yang dibuka) (buruk atau sesuai dengan kasus yang terjadi.
    b. Kompleksitas ruang: Ruang yang dipakai oleh algoritma, diukur dari ukuran fringe yang terbentuk.

Perbedaan strategi pencarian, menyebabkan munculnya beberapa pendekatan:

1. Pencarian buta (tanpa informasi): BFS, DFS
2. Pencarian dengan informasi: heuristik
3. Pencarian dengan batasan (constraints satisfaction)

4. Pencarian saling berlawanan (adversary)

Pencarian Buta

Terbagi atas dua algoritma utama, yaitu: Breadth First Search (BFS), dan Depth First Search (DFS).

2.4.4 Pohon Pencarian

Dari sebuah ruang pencarian, dapat dihilangkan semua siklus yang ada, untuk menghasilkan sebuah pohon pencarian. Terlebih dahulu perlu disampaikan beberapa terminologi terkait pohon pencarian. Pohon pencarian adalah sebuah data struktur yang terdiri atas sebuahsimpul induk utama, disebut root, tempat dimulainya pencarian. Setiap simpul dapat memili satu atau lebih simpul anak (child). Jika x adalah induk dari y, maka y adalah anak dari x.

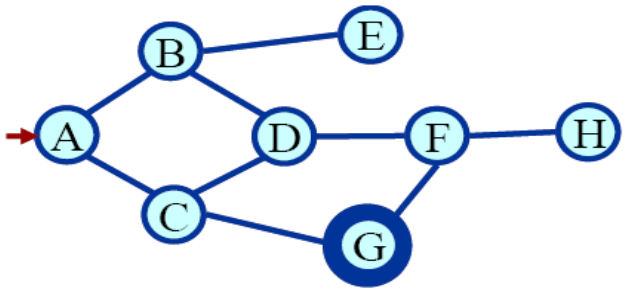


Figure 1: A State Space Graph

Jika diasumsikan bahwa hubungan dalam graf di atas dua arah (bidirectional),maka akan dihasilkan pohon pencarian sebagai berikut:

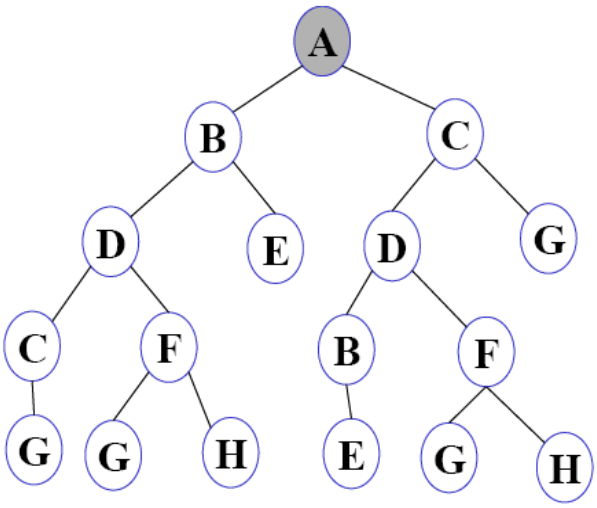


Figure 2: Search tree for the state space graph in Figure 25

Terminologi pohon pencarian:

Root Node: simpul dimulainya proses pencarian.

Leaf Node: simpul pada pohon pencarian yang tidak lagi memiliki anak.

Ancestor / Descendant: x adalah pendahulu (ancestor) dari y, jika x adalah parent dari y atau pendahulu dari parent y. Dalam hal ini, y adalah keturunan (descendant) dari x.

Branching factor: jumlah anak maksimum dari sebuah simpul, yang bukan berada pada leaf node.

Path: jalur lengkap adalah jalur yang dimulai dari root dan berakhir pada solusi. Jalur parsial adalah jalur yang tidak memiliki solusi.

Data struktur node (simpul):

1. Deskripsi keadaan
2. Pointer ke parent
3. Kedalaman simpul
4. Operator yang membuka simpul ini
5. Biaya total dari simpul awal sampai simpul ini

Simpul yang dibangkitkan / dibuka sesuai algoritma yang sedang dipakai akan ditempatkan pada sebuah data struktur OPEN atau fringe. Pada mulanya hanya simpul awal (root node) yang dibuka.

Pencarian dimulai dari root, algoritma kemudian mengambil sebuah simpul dari OPEN untuk dibuka semua anak-anaknya. Membuka sebuah simpul dari OPEN, akan menghasilkan data struktur CLOSED.

Sebuah solusi dari problem pencarian merupakan urutan operator, yang diasosiasikan dengan sebuah jalur dari simpul awal sampai tujuan. Biaya dari sebuah solusi merupakan penjumlah biaya total dari setiap keterhubungan (arcs) pada jalur solusi. Untuk ruang pencarian yang sangat besar, tentu tidak efektif untuk menampilkan semua simpul, lebih bermanfaat jika hanya membuka simpul pencarian yang mungkin akan berisi solusi(parsial dan implisit).

Proses pencarian akan menghasilkan pohon pencarian yang memiliki: root , dan leaf nodes (belum dibuka (di dalam fringe) atau yang tidak memiliki anak lagi (dead ends). Pohon pencarian bisa saja mengalami looping terus menerus (infinite), meskipun ruang pencariannya kecil.

Sebuah pencarian akan mengembalikan hasil sebuah jalur pencarian sampai ke tujuan yang dikehendaki. Menemukan jalur solusi penting bagi permasalahan, misalnya: rute terpendek, memecahkan 8-puzzle; namun ada juga problem yang hanya mementingkan hasil akhir, misalnya N-queens problem.

2.5 Breadth First Search (BFS)

2.5.1 Algoritma dasar BFS adalah:

```
Ambil fringe sebagai list berisi keadaan awal
Loop
   if fringe kosong return failure
   Node ← remove-first(fringe)
   if Node is a goal
   then return jalur dari keadaan awal sampai Node
   else buka semua suksesor Node, dan
   (satukan Nodes baru ke dalam fringe)
   tambahkan Nodes baru ke akhir fringe
End Loop
```

Dalam BFS, simpul baru diletakkan pada akhir fringe. Dengan demikian model antriannya adalah FIFO (first in first out). Simpul yang berada terlebih dahulu di dalam fringe, akan dibuka pertama pula. Dengan demikian pencariannya akan melebar, mengikuti pembukaan semua suksesor dari sebuah simpul. Implementasi yang cocok untuk BFS adalah dengan data struktur linked-list.

2.5.2 Ilustrasi BFS

Untuk pohon pencarian pada Figure 1, dapat disimulasikan algoritma BFS sebagai berikut:

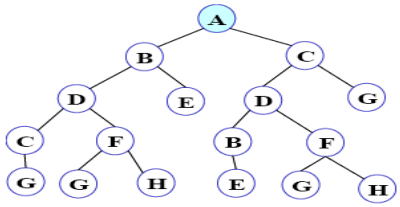


Figure 3

FRINGE: A

Mula-mula hanya ada satu simpul awal dalam fringe, A. Simpul A dikeluarkan dari fringe, dengan membuka semua suksesornya, yaitu B dan C

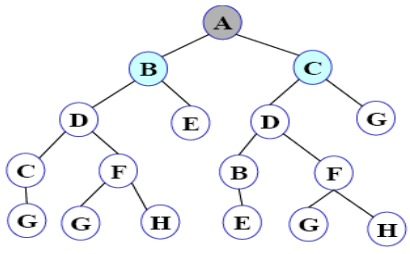


Figure 4

FRINGE: B C

Simpul B akan dibuka dan dibuang dari fringe, dengan membuka anak-anak B, yaitu D dan E, dengan meletakkannya di belakang C.

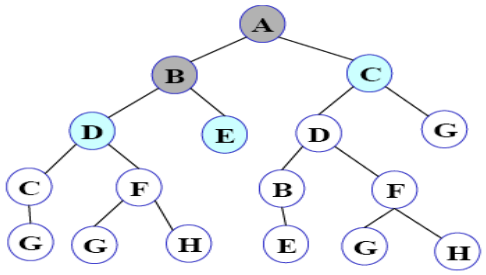


Figure 5

FRINGE: C D E

Simpul C dibuka, dan anak-anaknya, yaitu D dan G, diletakkan di belakang D, E.

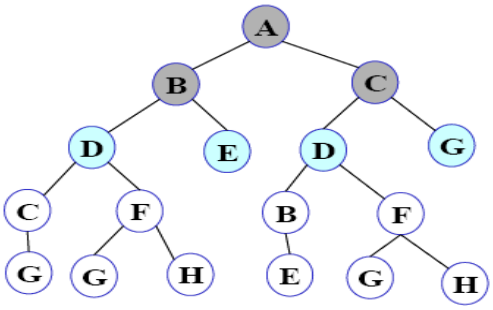


Figure 6

FRINGE: D E D G

Simpul D dikeluarkan dari fringe, dengan membuka anak-anak D, yaitu: C dan F, dan akan ditaruh di belakang E, D, G.

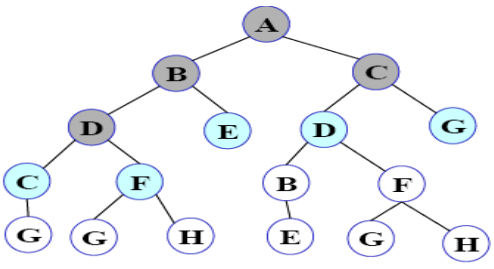
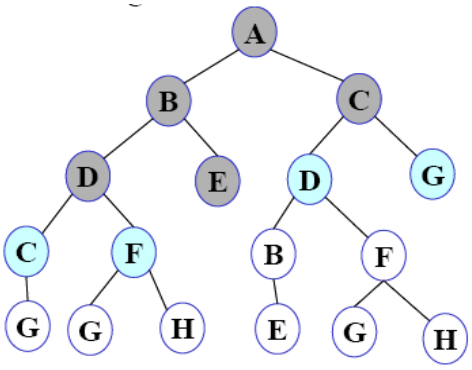


Figure 7

FRINGE: E D G C F

Simpul E, dibuang dari fringe, dan tidak memiliki anak.



FRINGE: D G C F

Simpul D, dibuka, dengan membuka simpul B dan F, kemudian ditaruh di belakang G, C, F.

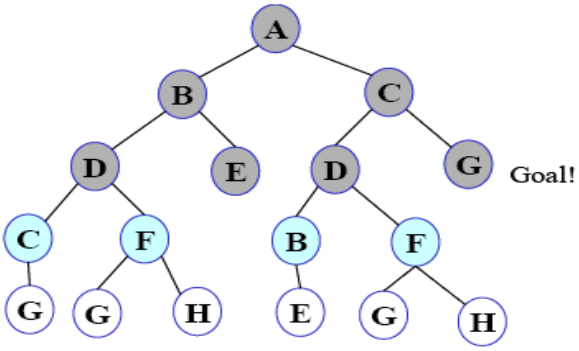


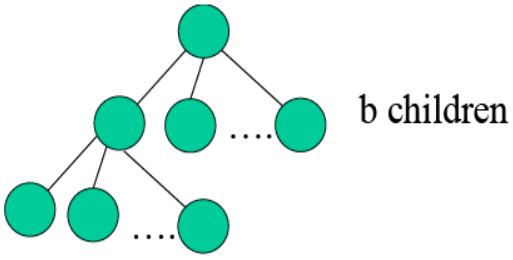
Figure 8

FRINGE: G C F B F

Simpul G dibuka, yang ternyata merupakan tujuan, algoritma berhenti.

2.5.3 Properti BFS

Ambil sebuah pohon pencarian:



Pohon memiliki branching factor = b. Asumsikan d sebagai kedalaman dari pohon, dan m adalah kedalaman tempat ditemukannya simpul pertama.

Algoritma pencarian BFS adalah:

Komplit: semua kemungkinan dalam satu tingkat kedalaman akan diekspan semuanya.

Optimal: (=admissable), jika semua operator memiliki biaya yang sama. Jika tidak, BFS akan menemukan solusi dengan jalur terpendek.

Memiliki ruang dan waktu eksponensial. Misalkan pada pohon pencarian b-ary di atas, kompleksitas ruang dan waktu adalah O(bd), dimana d adalah kedalaman solusi dan b adalah branching factor (jumlah anak dari setiap simpul).

Pohon pencarian lengkap untuk sebuah BFS, dengan kedalaman d, akan menghasilkan jumlah simpul:

$$1 + b + b^2 + \ldots + b^d = (b^{(d+1)} - 1)/(b-1)$$

BFS dapat digunakan secara efektif, jika ruang pencarian cukup kecil. Pencarian akan memakan waktu yang sangat lama kalau branching factor besar, dengan tingkat kedalaman besar. Dan jika diimplementasikan dengan komputer, akan segera memakan memori.

Keuntungan BFS:

Menemukan jalur solusi dengan jalur tersingkat, karena selalu mengambil lebar dari pohon pencarian.

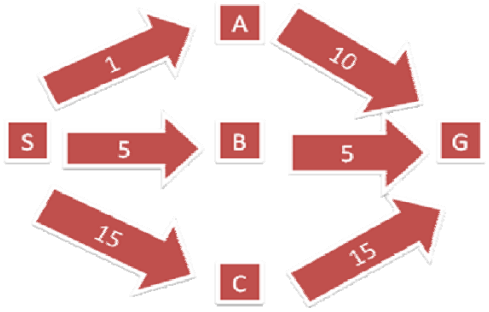
Kerugian BFS:

Memerlukan ruang dan waktu pencarian yang eksponensial pada kedalaman pencarian.

2.6 Uniform Cost Search

Algoritma ini diperkenalkan oleh Dijkstra (1959). Algoritma ini membuka simpul sesuai dengan urutan biaya secara menaik. Biasanya yang dijadikan sebagai biaya adalah jumlah biaya yang diperlukan dalam jalur pencarian. Dalam algoritma uniform cost, simpul yang baru akan diletakkan dalam OPEN sesuai dengan urutan biayanya. Hal ini akan menyebabkan bahwa simpul yang dipilih untuk dibuka adalah yang biayanya paling kecil (head dari list OPEN). Secara urutan menaik, yang didata adalah nilai g(n) = biaya jalur pencarian dari titik awal sampai node n.

Properti dari algoritma pencarian ini adalah: komplit, optimal / admissible, dan exponensial dalam kompleksiatas waktu dan ruang, $O(b^d)$.



Pada graf di samping, proses pencarian berlangsung sebagai berikut:

1. OPEN S (start)
2. OPEN A, B, C (cost = 1, 5, 15)
3. OPEN B, G, C (cost = 5, 11, 15)
4. OPEN G, G, C (cost = 10, 11, 15)
5. SOLUTION G (path S-B-G)

2.7 Depth First Search (DFS)

2.7.1 Algoritma

Algoritma dasar DFS adalah:

```
Ambil fringe sebuah list berisi initial state
Loop
   if fringe is empty return failure
      Node ← remove-first(fringe)
   if Node adalah tujuan
   then return jalur dari initial state ke Node
   else buka semua suksesor Node, dan
            satukan simpul baru ke dalam fringe
   tambahkan node baru ke depan, sebagai head dalam fringe
End Loop
```

Algoritma DFS selalu meletakkan simpul-simpul baru yang dibuka di depan list OPEN. Hal ini menyebabkan pembukaan dilakukan secara kedalaman suatu simpul. Jadi simpul dalam OPEN mengikuti urutan LIFO (Last In First Out). OPEN akan diimplementasikan secara data struktur stack (tumpukan).

2.7.2 Ilustrasi DFS

Proses pencarian dengan algoritma DFS dapat dilihat dalam contoh sebagai berikut:

Graf pencarian yang ada adalah

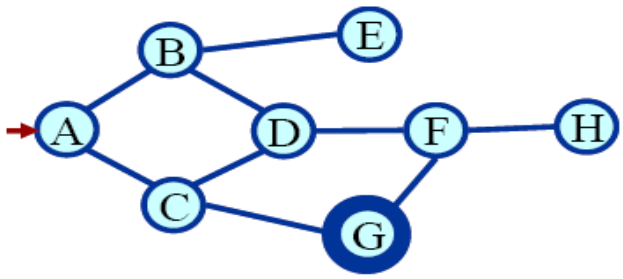


Figure 10

Dari graf perlu untuk diubah menjadi pohon pencarian sebagai berikut:

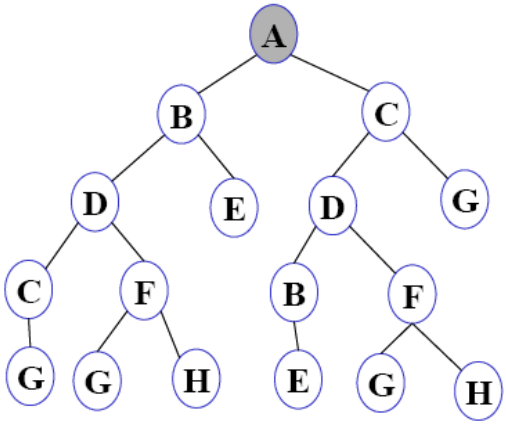
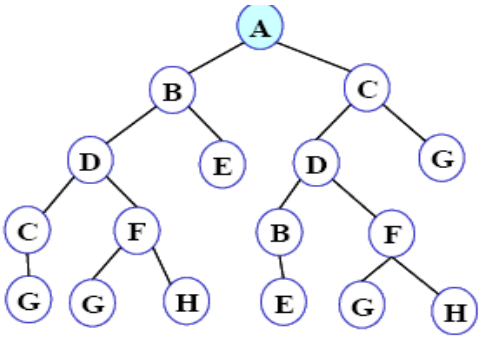


Figure 11: Search tree for the state space graph in Figure 34

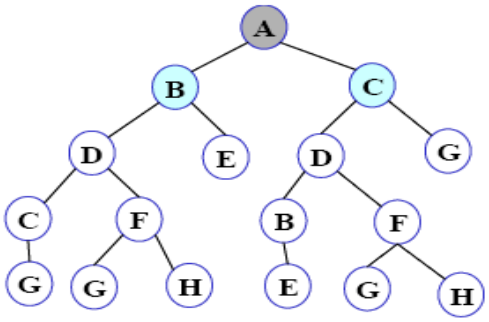
Initial state berada pada node A.



FRINGE: A

Figure 12

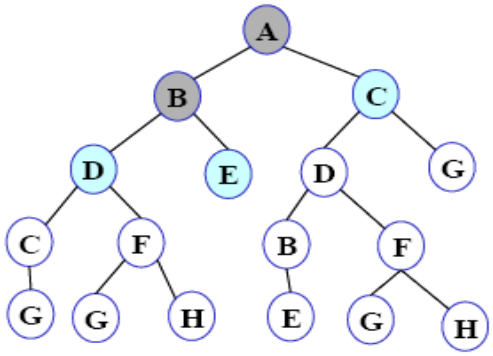
Karena A bukan tujuan, maka anak-anak A dibuka.



FRINGE: B C

Figure 13

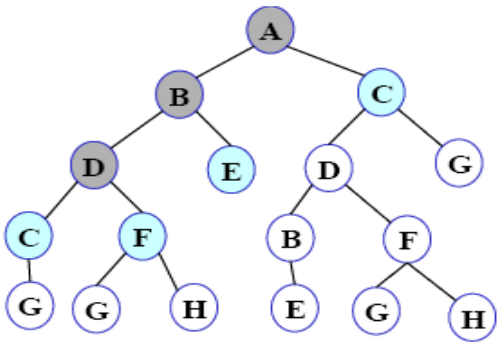
Simpul B dibuka, karena bukan tujuan. Simpul baru D dan E, diletakkan di depan fringe.



FRINGE: D E C

Figure 14

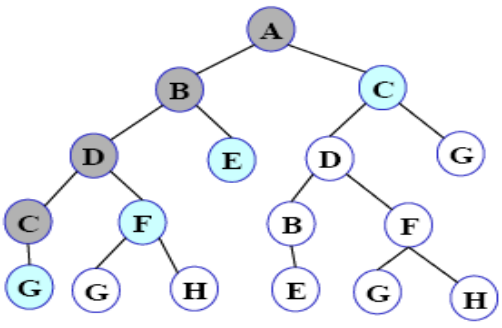
Karena D bukan tujuan, maka harus dibuka.



FRINGE: C F E C

Figure 15

Simpul C dibuka, karena bukan merupakan tujuan.



FRINGE: G F E C

Figure 16

Simpul G diletakkan di depan fringe, dan karena G adalah tujuan, maka pencarian dihentikan, dengan jalur solusi A-B-D-C-G.

2.7.3 Properti DFS

Algoritma DFS memerlukan waktu eksekusi yang eksponensial. Jika N adalah kedalaman maksimum dalam pohon pencarian, maka dalam keadaan terburuk algoritma akan memerlukan waktu $O(b^d)$. Namun untuk kompleksitas ruang / memori, berkembang secara linear pada kedalaman pohon pencarian, O(bN).

Perlu diperhatikan bahwa kebutuhan waktu dalam proses pencarian dengan DFS, sejalan dengan kedalaman maksimum pohon pencarian. Jika kedalaman pohon tidak terbatas, maka dimungkinkan bawah algoritma tidak akan berhenti. Hal ini dapat terjadi jika ruang pencarian tidak terbatas, atau jika ruang pencarian mengandung siklus keadaan. Dengan demikian algoritma DFS tidak menunjukkan sifat komplit, hanya sebagian ruang pencarian yang ditempuh.

2.7.4 Depth Limited Search

Sebuah variasi DFS untuk mencegah ketidaklengkapan pencarian, dapat dilakukan dengan cara memberikan batas kedalaman. Simpul akan dibuka jika belum mencapai batas maksimum kedalaman pohon pencarian. Algoritma ini dikenal dengan depth-limited search.

```
Ambil fringe yang memiliki initial state
Loop
   if fringe kosong return failure
      Node ← remove-first(fringe)
   if Node adalah tujuan
   then
      return jalur dari initial state sampai Node
   else
      if kedalaman Node = limit return cutoff
      else tambahkan node yang baru dibuka ke depan fringe
End Loop
```

2.7.5 Depth First Iterative Deepening (DFID)

Algoritma ini digunakan untuk membatasi kedalaman pencarian. Mula-mula dilakukan DFS pada tingkat / level 0 saja (seolah-oleh tidak mempunyai suksesor), jika solusi tidak ditemukan, lakukan DFS pada tingkat berikutnya, dst.

```
until ditemukan solusi do
   DFS dengan kedalaman pada level c
   c = c+1
```

Keuntungan algoritma ini:

- Kebutuhan memori yang linear, seperti DFS
- Garansi dicapainya simpul tujuan, dengan kedalaman minimal (=1 tingkat saja)

Proses perjalanan algoritma ini dapat dilihat pada gambar berikut. Terlihat bahwa setiap kali pencarian dilanjutkan, tingkat kedalaman bertambah satu.

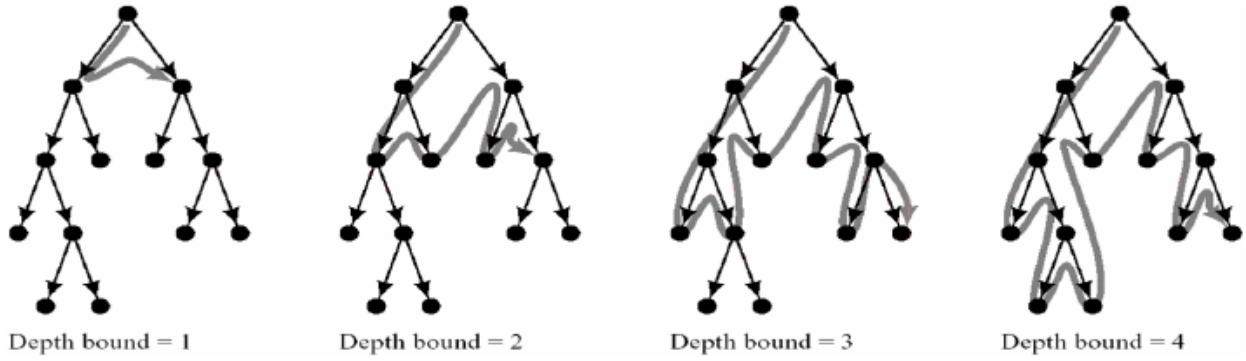


Figure 18: Depth First Iterative Deepening

Properti DFID

Untuk tingkat kedalaman yang besar, perbandingan jumlah simpul yang dikembangkan oleh DFID jika dibandingkan dengan DFS adalah b/(b-1). Misalnya untuk branching factor sejumlah 10 dan pencapaian tujuan yang dalam, diperlukan 11% lebih banyak jumlah simpul dalam DFID dibandingkan dengan BFS.

Dengan demikian, maka algoritma DFID:

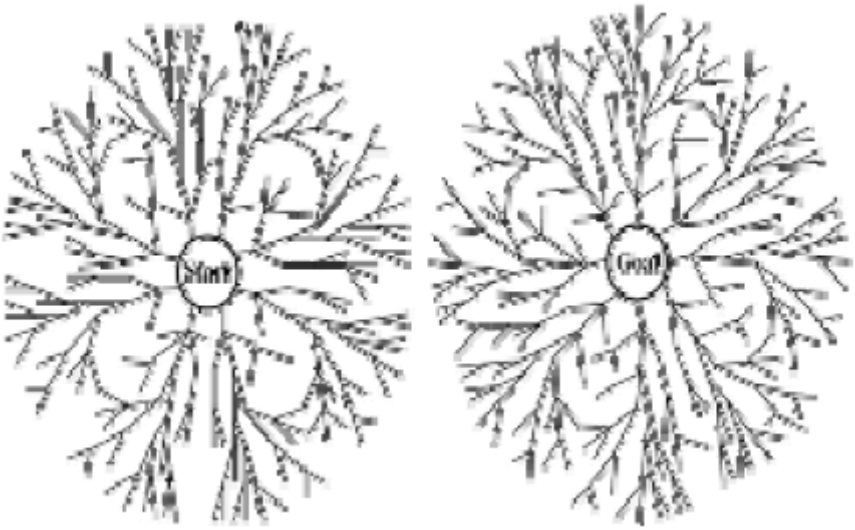
- Komplit
- Optimal / Admissible, jika semua operator memiliki biaya jalur yang sama. Jika tidak, maka tidak optimal, namun memberikan garansi penemuan solusi yang lebih pendek dibanding DFS.
- Untuk kompleksitas waktu pencarian sedikit lebih buruk dari BFS atau DFS, karen simpul yang dekat ke puncak akan dibuka beberapa kali, tetapi karena lebih banyak simpul yang berada di dasar pohon pencarian, maka kompleksitas waktu masih dalam orde eksponensial $O(b^d)$. Jika branching factor adalah b, dan kedalaman solusi berada pada d, maka simpul pada kedalaman d akan dibuka sebanyak sekali, pada (d-1) akan dibuka dua kali, dst, sehingga $b^d + 2b^{(d-1)} + ... + db <= b^d / (1 - 1/b)^2 = O(b^d)$.
- Untuk kompleksitas ruang, pada orde O(bd), seperti DFS.

DFID mengkombinasikan keuntungan BFS (pada sisi kelengkapannya) dan DFS (pada sisi efektivitas ruang dan jalur pencarian). Algoritma ini akan dapat digunakan sebagai strategi untuk pencarian yang memerlukan ruang pencarian sangat besar, dengan kedalaman yang tidak dapat ditebak sebelumnya.

Ada pula teknik sejenis, yang disebut dengan iterative broadening (IB), yang digunakan ketika ada banyak simpul tujuan. Algoritma IB akan membuka setiap kali satu anak dari setiap simpul. Pada iterasi kedua, dua anak, dan pada iterasi ke-i akan membuka I anak.

Pencarian Dua Arah (Bi-directional)

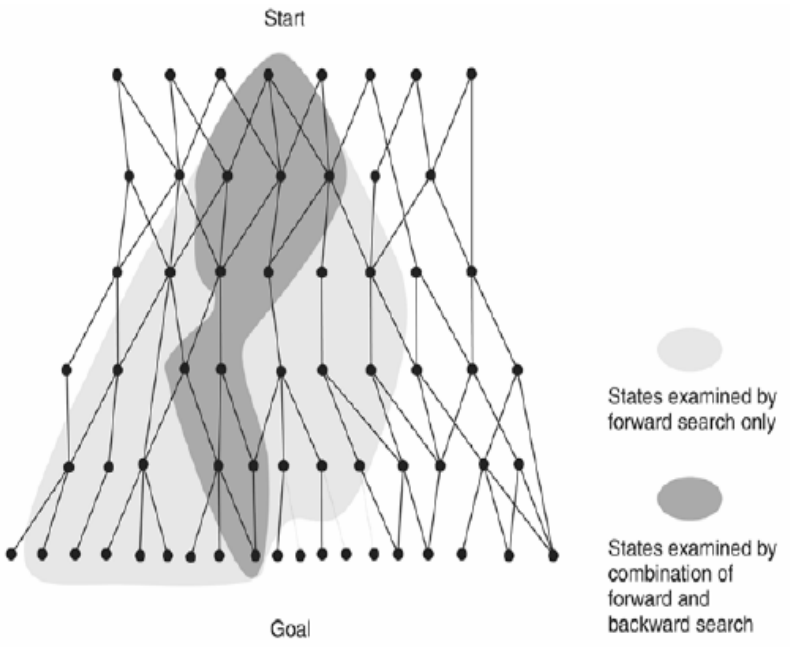
Asumsikan bahwa problem dapat dipecahkan dengan pencarian yang dua arah, yaitu jika ada operator yang membawa keadaan A ke B, dan ada pula yang membawa B ke A. Contoh-contoh problem yang dapat didekati dengan cara ini, misalnya: 8-puzzle, 15-puzzle, planning, dsb. Namun ada kalanya ada problem definisi yang tidak memungkinkan cara ini, seperti problem teko air. Jika seandainya ada jalur yang dapat dibalik arah pencariannya, maka sebenarnya pencarian akan dapat dilakukan secara terbalik, satu memulai dari keadaan awal sampai tujuan, yang lainnya memulai dari keadaan akhir / tujuan hingga mencapai keadaan awal. Bagaimana cara yang terbaik untuk mencari mulai dari tujuan? Yang perlu diperhatikan disini adalah bagaimana dari sebuah simpul dapat dicari keadaan yang sebelumnya (predecessor).



Algoritma: pencarian dua arah, melakukan secara bergantian, dari keadaan awal ke tujuan dan dari tujuan ke keadaan awal, dan akan berhenti jika terjadai interseksi (ada keadaan yang sama).

Setiap kali pencarian dilakukan, baik dari keadaan awal maupun dari tujuan, sebuah algoritma (yang sama) harus dipilih. Jika dicapai sebuah keadaan yang sama, maka solusi ditemukan, namun harus dipastikan bahwa keadaan tersebut harus ditemukan, baik dengan pencarian dari tujuan maupun dari keadaan awal.

Terkadang pencarian dua arah akan menemukan solusi lebih cepat. Hal ini dikarenakan pencarian seolah terbagi dalam dua sisi pencarian, yang pada akhirnya bertemu di tengah. Mekanisme ini dapat digambarkan sebagai berikut:



Pencarian dua arah akan berhasil dengan baik, jika ada simpul awal dan akhir yang spesifik (unik). Algoritma ini akan sulit diterapkan, jika ada lebih dari satu simpul tujuan.

Kompleksitas Waktu dan Ruang

Anggaplah sebuah pohon pencarian dengan branching factor b. Jika tujuan dapat dicapai dengan d langkah dari keadaan awal, maka BFS akan melakukan pembukaan jumlah simpul sebanyak $O(b^d)$. Jika dilakukan pencarian dengan dua arah, maka keadaan solusi, mungkin akan ditemukan ditengah-tengah, dengan kedalaman d/2. Dengan demikian maka kompleksitas pencarian dua arah menjadi $O(b^{d/2})$. Perhatikan pula bahwa paling tidak satu dari keadaan yang sama akan disimpan, sehingga ruang kompleksitas pencarian menjadi juga $O(b^{d/2})$.

Membandingkan Strategi Pencarian

Tabel berikut memberikan ringkasan performa algoritma pencarian buta yang telah dituliskan sebelumnya, sebagai berikut:

	Breadth first	**Depth first**	**Iterative deepening**	**Bidirectional (if applicable)**
Time	b^d	b^d	b^d	$b^{d/2}$
Space	b^d	bm	bd	$b^{d/2}$
Optimum?	Yes	No	Yes	Yes
Complete?	Yes	No	Yes	Yes

Pencarian di dalam Graph

Jika pencarian dilakukan bukan melalui sebuah pohon, namun dalam sebuah graf, maka akan dapat terjadi keterhubungan keadaan yang mengacu pada hal yang sama (repetitive states).

Untuk meyakinkan bahwa hanya simpul yang belum dibukalah yang akan diekspansi, diperlukan adanya data struktur CLOSED, yang berisi informasi semua simpul yang pernah dibuka (dan kalau bisa tidak perlu dibuka lagi). Selengkapnya dapat digambarkan sebagai berikut:

```
Let fringebe a list containing the initial state
Let closedbe initially empty
Loop
if fringeis empty return failure
   Node ← remove-first (fringe)
   if Node is a goal
   then
       return the path from initial state to Node
   else put Node in closed
       generate all successors of Node S
       for all nodes m in S
       if m is not in fringe or closed
           merge m into fringe
End Loop
```

Algoritma graf ini agak boros. Selain karena memerlukan list CLOSED, algoritma ini melakukan pemeriksaan setiap simpul pada OPEN dan CLOSED. Perbaikan yang dapat dilakukan adalah dengan membuat index pada simpul, sehingga dapat mendapat keadaan simpul tanpa harus melalui data struktur CLOSED.

Ada strategi-strategi yang dapat dikembangkan sendiri, tanpa harus memodifikasi algoritma pencarian. Misalnya dengan tidak kembali kepada keadaan pada tahap sebelumnya. Strategi seperti ini akan banyak meningkatkan performa algoritma, misalnya untuk menghindari pengulangan keadaan pada 15-puzzle.

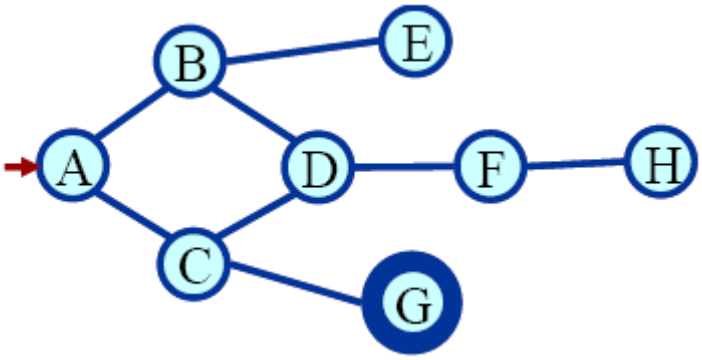
Strategi lainnya adalah misalnya dengan melakukan pemeriksaan cycle pada path. Cycle dapat menyebabkan terjadinya perulangan state.

Strategi berikutnya adalah dengan menghindari membuka state yang sudah pernah dibuka sebelumnya. Dengan usaha untuk menghindari state lama, maka kompleksitas pohon dan waktu pencarian akan terpangkas.

Strategi mana yang akan diterapkan, tentu akan ditentukan dari seberapa besar perulangan yang terjadi.

PERTANYAAN:

1. Ambil graf sebagai berikut sebagai sebagai sebuah ruang pencarian:



Dimulai dari simpul A dengan tujuan G, lakukan algoritma DSF. Jelasnkan apa isi dari fringe, dan bagaimana ekspansi simpul dilakukan. Asumsikan bahwa simpul dengan urutan alfabet yang lebih awal dibuka terlebih dahulu, jika ada lebih dari satu simpul yang dapat dibuka bersamaan.

2. Anggap Anda memiliki tabel keadaan dan biaya jalur sebagai berikut:

State	next	cost
A	B	4
A	C	1
B	D	3
B	E	8
C	C	0
C	D	2
C	F	6
D	C	2
D	E	4
E	G	2
F	G	8

a. Gambarlah ruang keadaan yang ada.

b. Asumsikan keadaan awal adalah A, dan tujuan adalah G. Tunjukkan bagaimana proses pencarian terjadi untuk setiap algoritma berikut, gunakan pohon pencarian yang sama: BFS, DFS, Uniform Cost, DFID. Untuk setiap algoritma tampilkan isi fringe, dan bagaimana urutan pembukaan simpul terjadi. Hitunglah pula biaya jalurnya.