Aula 06 – Resolução de problemas

22705/1001336 - Inteligência Artificial
2019/1 - Turma A
Prof. Dr. Murilo Naldi

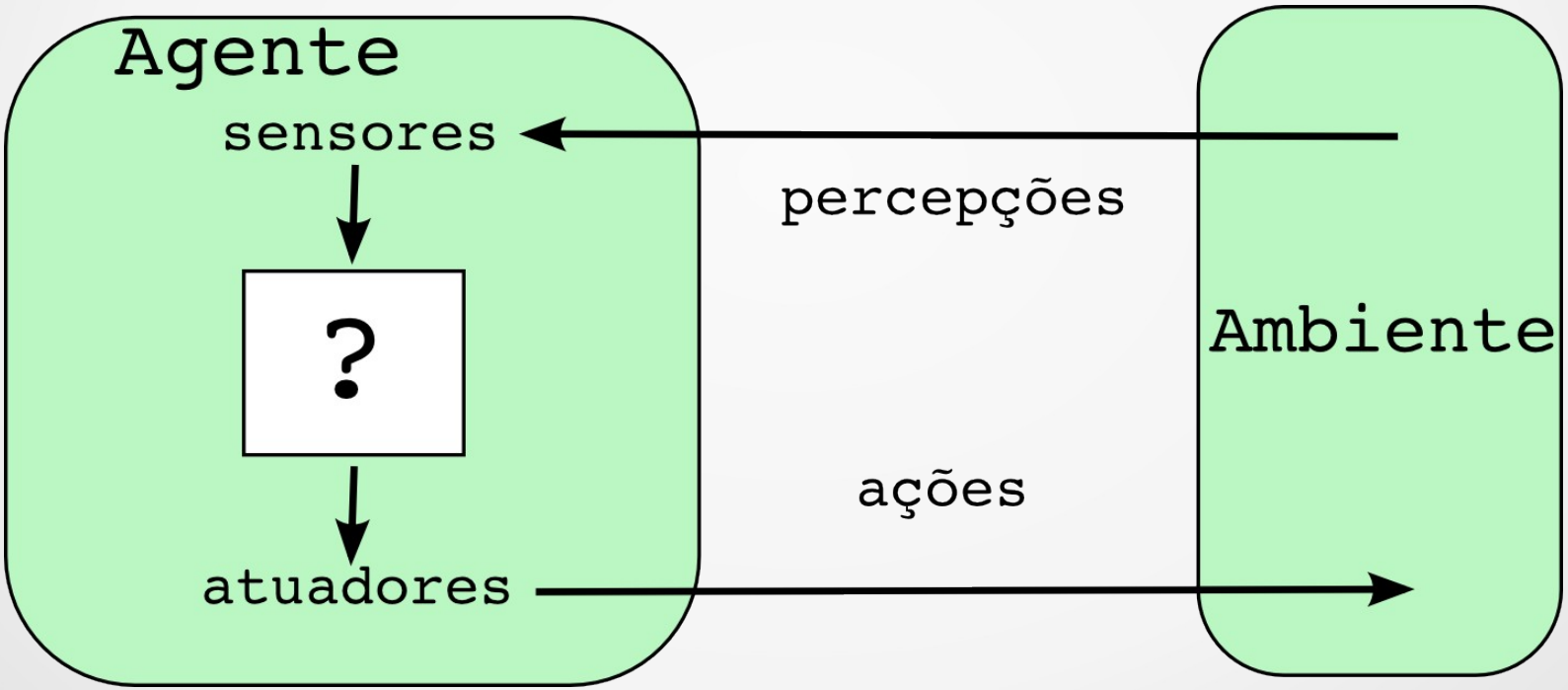
naldi@dc.ufscar.br

Agradecimentos

- Agradecimentos pela base do material utilizado nesta aula foi cedido ou adaptado do material dos professores Andréia Bonfante, Heloísa Camargo, Ricardo Campello e Ricardo Cerri.

Agente Inteligente

- Agente é tudo capaz de perceber seu ambiente e agir sobre esse ambiente.



Tomada de decisão

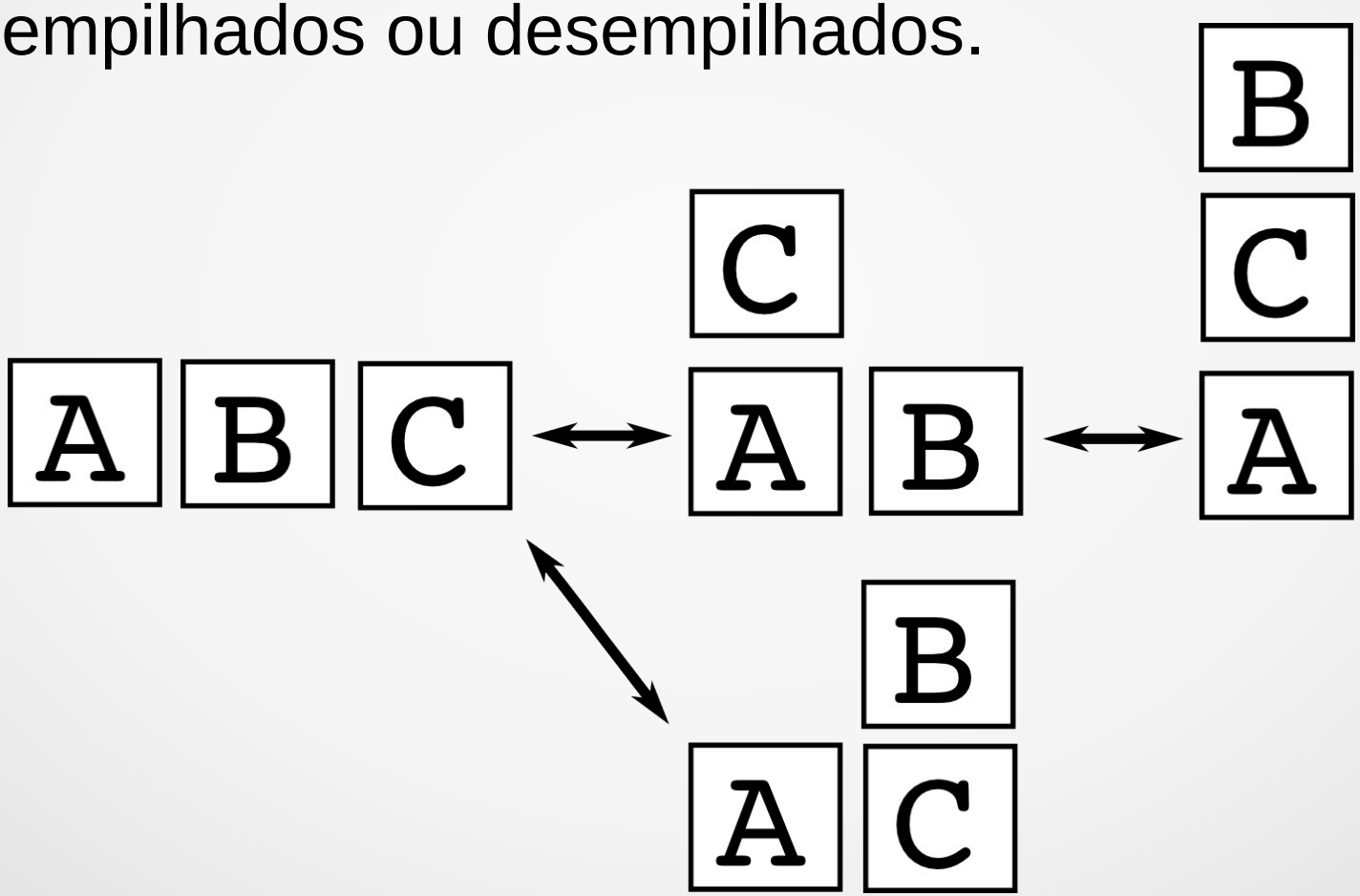
- Dada uma percepção do ambiente, qual ação tomar?
  - Existem diversas formas
    - Motor de inferência lógica (Prolog)
    - Modelo de um classificador
    - … dentre outros.
- Podemos utilizar o Prolog para solucionar problemas de estado, em que:
  - Estado é definido pela percepção do ambiente
  - Ação modifica o ambiente

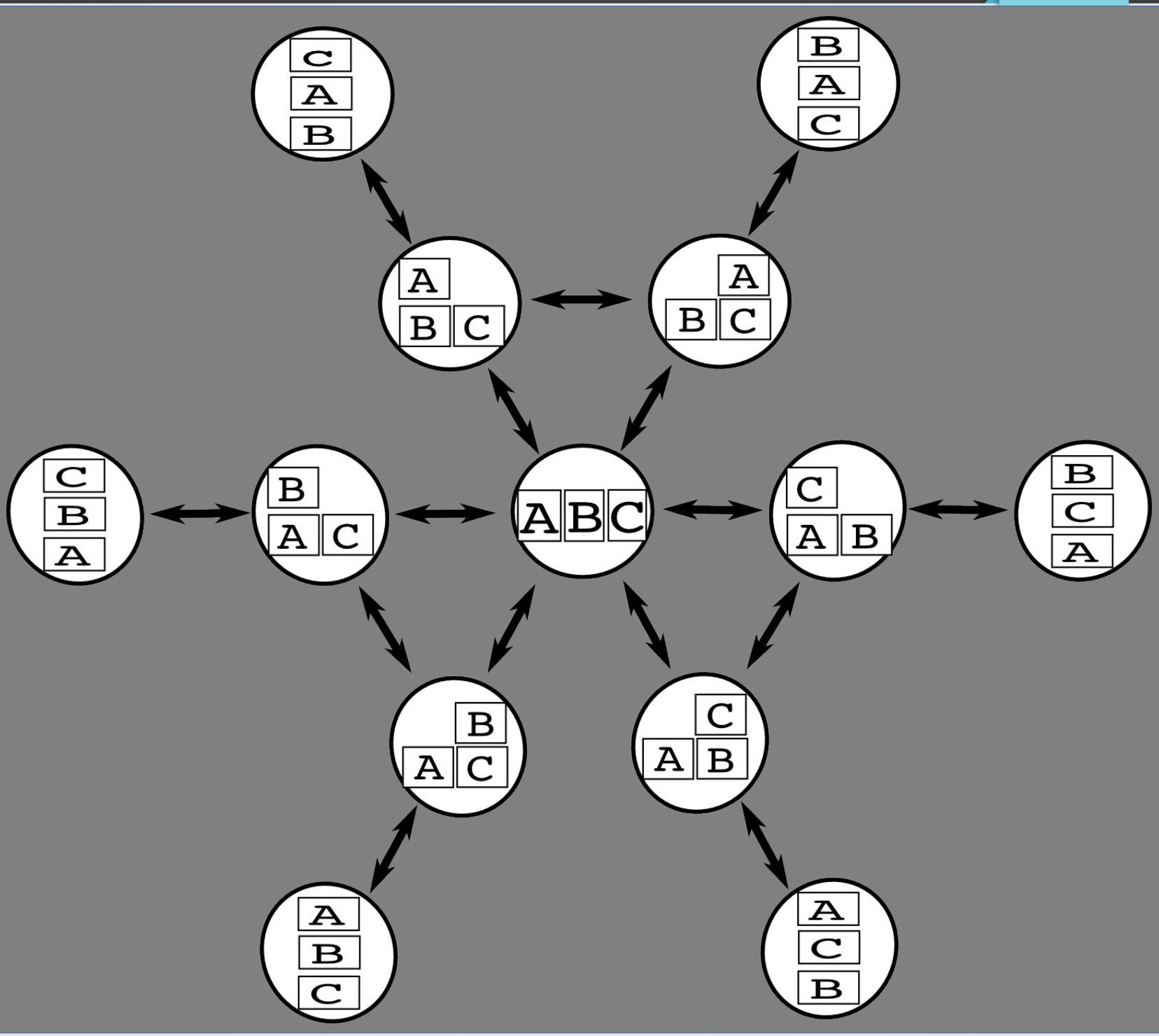
Problemas de estado

- Problemas de estado podem ser representados em um espaço de estados
- Um espaço de estados pode ser visto como um grafo em que:
  - Os nós são os estados do problema
  - As arestas são as ações que causam transições entre esses estados
- O agente deve aplicar as ações necessárias para modificar o ambiente até que ele consiga encontrar o estado objetivo.

Exemplo

- Problema de blocos: três blocos A, B e C podem ser empilhados ou desempilhados.




C
A
B
B
A
C
A
B C
A
B C
C
B
A
B
A C
A B C
C
A B
B
C
A
B
A C
C
A B
A
B
C
A
C
B

Definir problema de estados

- Em essência, grafo é apenas uma ilustração
  - Não é prático gerar um grafo do espaço de estados completo para cada problema
- Portanto, os problemas geralmente são representados pelos estados e transições (ações).
  - Estados são representados por objetos
    - Simples ou compostos (funções, listas, etc.)
  - Transições são representadas por ações
    - Predicados de relações entre estados

Definir estados

- Em Prolog, são objetos que contém todas as informações necessárias para definir um estado (ambiente) do problema
- Exemplo:
  - No problema de blocos, temos até três pilhas
    - cuja ordem horizontal é irrelevante
  - As pilhas por sua vez são representadas por listas, dentro de uma lista
    - [[ ],[c,a,b],[ ]] ou [[ ],[a,b],[c]] ou [[a],[b],[c]]

Definir transições

- Transições: definidas por meio de ações
- De forma geral, estabelecemos a relação de sucessor por meio de:
  - s(X,Y) % X é sucedido por Y
  - s(X,Y,Custo) % X é sucedido por Y com % custo = Custo
- Nestes casos, Y é sucessor de X e X é predecessor de Y
- Qualquer ação de mude o estado do problema é uma ação de transição.

Exemplo transições

- A transição do problema dos blocos consiste em remover um bloco do topo de uma pilha e colocar em outra

- Para isso, iremos utilizar a remoção de lista m Prolog (dada na aula anterior):

  retirar_elemento(Elem,[Elem|Cauda],Cauda).

  retirar_elemento(Elem,[Elem1|Cauda],[Elem1|Cauda1]) :- retirar_elemento(Elem,Cauda,Cauda1).

Exemplo transições

- No problema dos blocos, uma situação é sucessora de X se houver duas pilhas em X, L1= [C1|P1] e L2, tais que o bloco do topo C1 (cabeça) de L1 é movido para L2
- Em Prolog, desmontamos o estado X e usamos as pilhas para montar seu sucessor, o estado Y

```
s(X, [P1, [C1|L2] | Resto ]):-
retirar_elemento([C1|P1], X, Outros), %acha P1
retirar_elemento(L2, Outros, Resto),%acha L2
not((P1 = [ ], L2 = [ ])). %evite formar pilha igual
```

Exemplo consulta

? – s([[a,b],[c],[ ]], [[b],[a,c],[ ]]).

Inferência (unificações):

C1 = ?

P1 = ?

Outros = ?

L2 = ?

Resto = ?

Resultado: ?

Exemplo consulta

? – s([[a,b],[c],[ ]], [[b],[a,c],[ ]]).

Inferência (unificações):

C1 = a

P1 = ?

Outros = ?

L2 = ?

Resto = ?

Resultado: ?

Exemplo consulta

? – s([[a,b],[c],[ ]], [[b],[a,c],[ ]]).

Inferência (unificações):

C1 = a

P1 = [b]

Outros = ?

L2 = ?

Resto = ?

Resultado: ?

Exemplo consulta

? – s([[a,b],[c],[ ]], [[b],[a,c],[ ]]).

Inferência (unificações):

C1 = a

P1 = [b]

Outros = [[c],[ ]]

L2 = ?

Resto = ?

Resultado: ?

Exemplo consulta

? – s([[a,b],[c],[ ]], [[b],[a,c],[ ]]).

Inferência (unificações):

C1 = a

P1 = [b]

Outros = [[c],[ ]]

L2 = [c]

Resto = ?

Resultado: ?

Exemplo consulta

? – s([[a,b],[c],[ ]], [[b],[a,c],[ ]]).

Inferência (unificações):

C1 = a

P1 = [b]

Outros = [[c],[ ]]

L2 = [c]

Resto = [[ ]]

Resultado: ?

Exemplo consulta

? – s([[a,b],[c],[ ]], [[b],[a,c],[ ]]).

Inferência (unificações):

C1 = a

P1 = [b]

Outros = [[c],[ ]]

L2 = [c]

Resto = [[ ]]

Resultado: true

Exemplo de consulta

? – s([[a,b],[c],[ ]], X).

X= [[b], [a, c], [ ]] ;

Exemplo de consulta

? – s([[a,b],[c],[ ]], X).

X= [[b], [a, c], [ ]] ;

X= [[b], [a], [c]] ;

Exemplo de consulta

? – s([[a,b],[c],[ ]], X).

X= [[b], [a, c], [ ]] ;

X= [[b], [a], [c]] ;

X= [[ ], [c, a, b], [ ]] ;

false %não encontra mais soluções

Outra forma?

- Da forma que definimos s(X,Y), Y deve obedecer uma ordem das pilhas: pilha que perde, pilha que recebe, resto.
- Podemos generalizar melhor essa transição, de forma que a ordem das pilhas de Y não importe?

Outra forma?

- Da forma que definimos s(X,Y), Y deve obedecer uma ordem das pilhas: pilha que perde, pilha que recebe, resto.
- Podemos generalizar melhor essa transição, de forma que a ordem das pilhas de Y não importe?

```
s(X,Y):-
retirar_elemento([C1|P1], X, Outros),
retirar_elemento(L2, Outros, [L3|_]),
not((P1 = [ ], L2 = [ ])),
pertence([C1|L2],Y), pertence(P1,Y), pertence(L3,Y), conta(Y,3).
```

Outra forma?

- Quais as vantagens e desvantagens dessa nova regra de transição em relação a antiga? (teste e responda)

```
s(X,Y):-
retirar_elemento([C1|P1], X, Outros),
retirar_elemento(L2, Outros, [L3|_]),
not((P1 = [ ], L2 = [ ])),
pertence([C1|L2],Y), pertence(P1,Y), pertence(L3,Y), conta(Y,3).
```

Definindo o problema

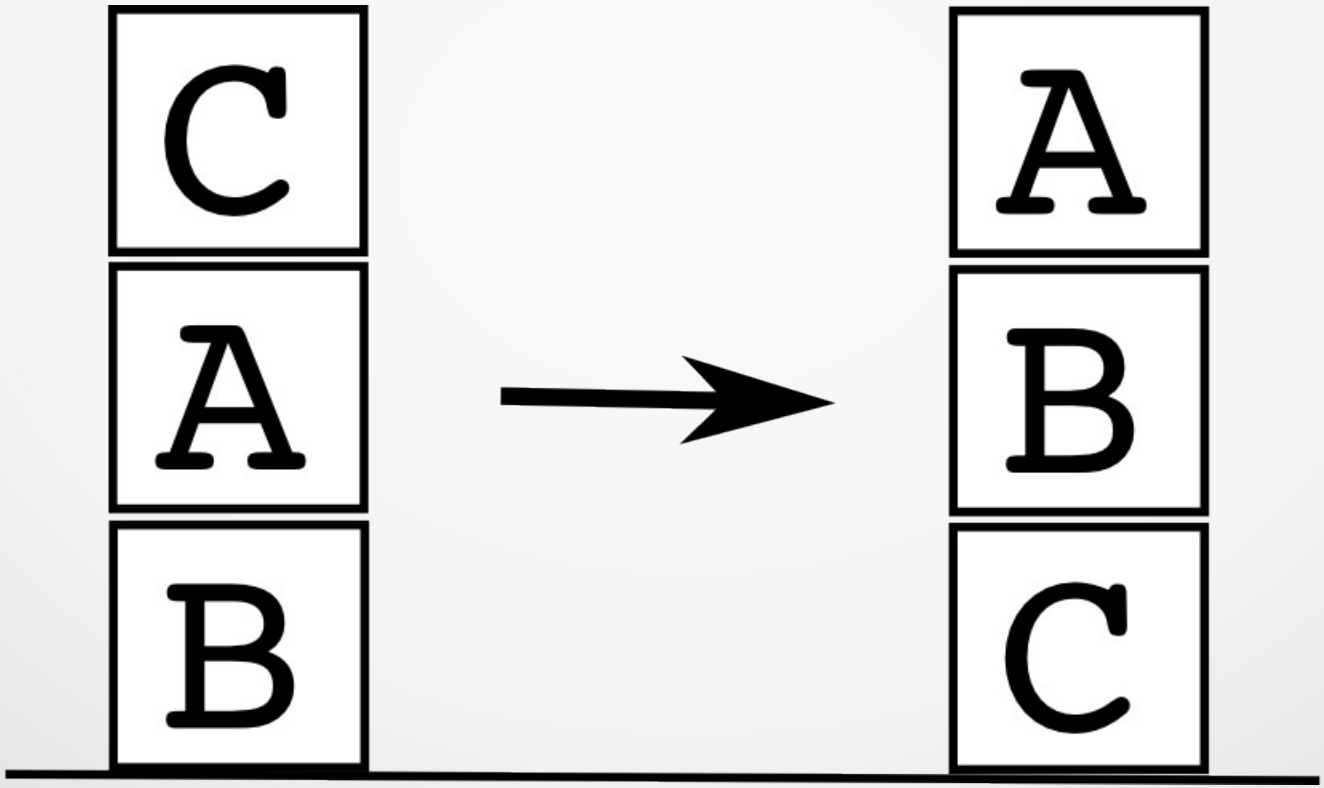
- Uma vez que definimos os estados e suas transições, podemos tentar responder a seguinte questão:
  - Dado um estado inicial, quais ações levam ao estado objetivo?
- Ou seja, desejamos descobrir um conjunto de estados ou/e ações que nos levam a esse objetivo!
  - O que é equivalente a buscar um caminho que vai do estado atual até esse objetivo!

Tipos de problemas

- Problemas podem possuir:
  - Um único estado inicial OU
  - Múltiplos estados iniciais
    - Nesse caso, escolhe-se um deles a cada execução.
  - Um único estado final OU
  - Múltiplos estados finais
    - Encontrar um caminho para qualquer estado final OU
    - Encontrar o melhor caminho para um dos estados finais

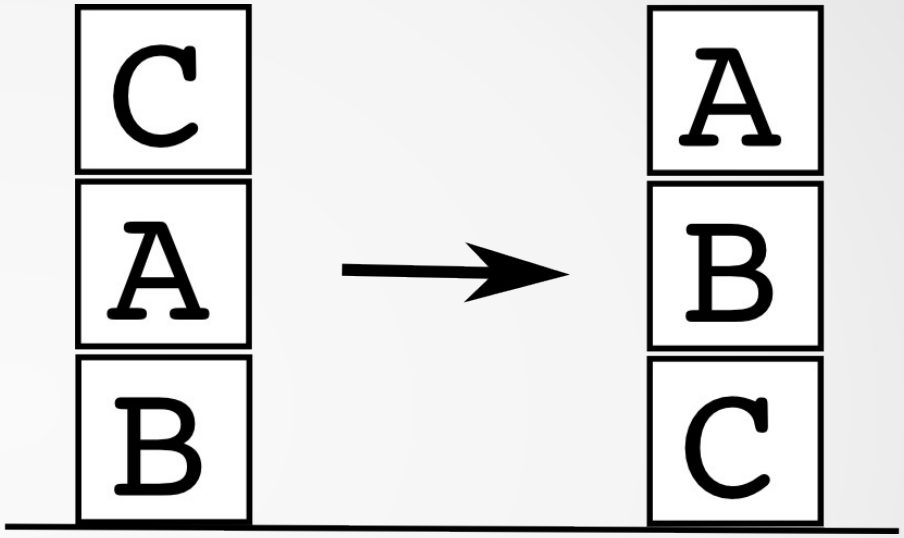
Problema dos blocos...

- Podemos definir os estados inicial e final do nosso problema de blocos da seguinte forma:



Definição do problema

- Estado inicial
  - [[c,a,b],[ ],[ ]]
- Estados objetivos possíveis
  - [[a,b,c],[ ],[ ]]
  - [[ ],[a,b,c],[ ]]
  - [[ ],[ ],[a,b,c]]



- Os estados objetivos podem ser definidos como:
  - meta(X) :- pertence([a,b,c],X)
  - em que pertence verifica se um elemento pertence a uma lista (aula anterior)

E agora?

- Pronto, já sabemos como definir um espaço de estados e como definir um estado inicial e um estado final (ou nó no caso do grafo)
- Só falta saber como encontrar o caminho!?
- Diferentes estratégias podem ser utilizadas para escolher caminhos

Buscas

- Buscas são estratégias para encontrar caminhos entre o estado inicial e o final de um problema em um espaço de estados.
- Um mesmo problema pode ser resolvido por diferentes tipos de busca.
- Cabe ao programador saber escolher qual a mais apropriada.

Como avaliar?

- Métodos de busca são avaliados por meio das seguintes características:
  - Completeza: o algoritmo sempre encontra uma solução se ela existe?
  - Complexidade de tempo: número de estados visitados/expandidos
  - Complexidade de espaço: número máximo de estados na memória
  - Admissibilidade: um algoritmo é admissível se ele garante encontrar uma solução ótima, quando ela existe.

Conceitos importantes

- Os espaço de estados pode ser divido em três partes durante a busca:
  - Espaço explorado (E): consiste no conjunto de estados explorados durante a busca
  - Fronteira (F): conjunto de estados sucessores aos explorados que não foram explorados ainda
  - Espaço não explorado: estados que existem, mas a busca não explorou e nem são sucessores de estados explorados

Métodos de busca

- Métodos de busca podem ser divididos em:

- Busca cega
  - (a seguir)
- Busca heurística
  - (a ser visto)

Busca Cega

- Também conhecida como busca não informada
  - Baseada somente no estado inicial
- É possível apenas conhecer quais estados são sucessores do estado atual
- Não há nenhuma informação adicional (ex.: o número de estados ou custo) sobre o caminho até a meta
- Importante para problemas nos quais não há informação adicional ao conjunto de estados explorados!

Alguns tipos de busca cega

- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Profundidade Iterativa

Busca em largura

- Consiste em explorar pelo(s) objetivo(s) na ordem em que os estados são apresentados ao mecanismo de busca
- Novos estados são inseridos no final da lista de estados de fronteira (F)
- Para evitar repetição de caminhos, usa lista de espaços explorados (E)
- Tem esse nome porque quando aplicado em árvores (grafos conexos e acíclicos), a busca é feita lateralmente da esquerda para a direita!!

Algoritmo BL

1 Inserir os nós iniciais na lista de busca F % fronteira

2 Se F é vazio

2.1 Então a busca não foi bem sucedida

3 Senão seja n o primeiro estado de F

3.1 Se n é um estado meta então

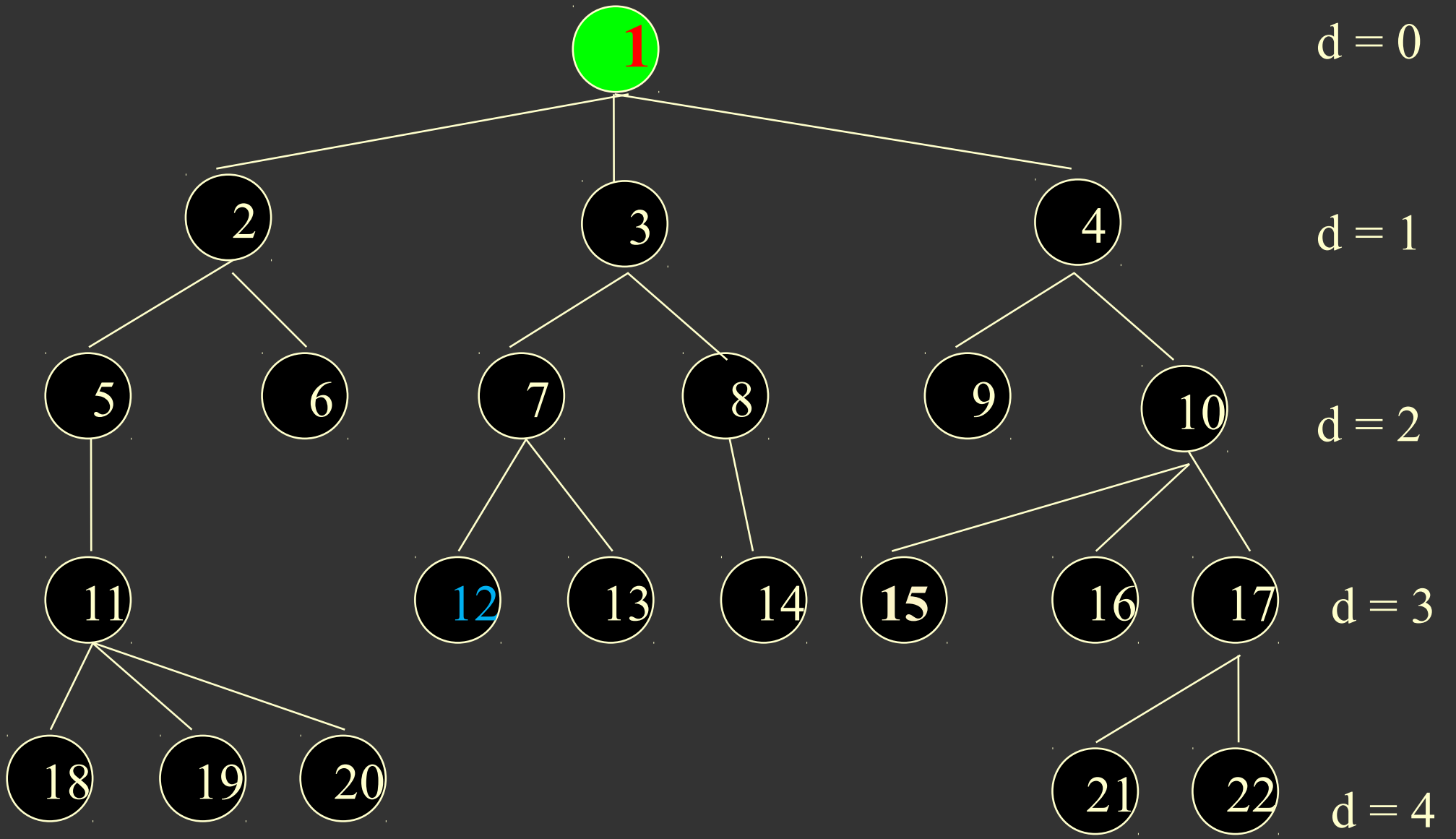
3.1.1 Retornar n

3.2 Senão

3.2.1 Remover n de F e inserir em E % explorados

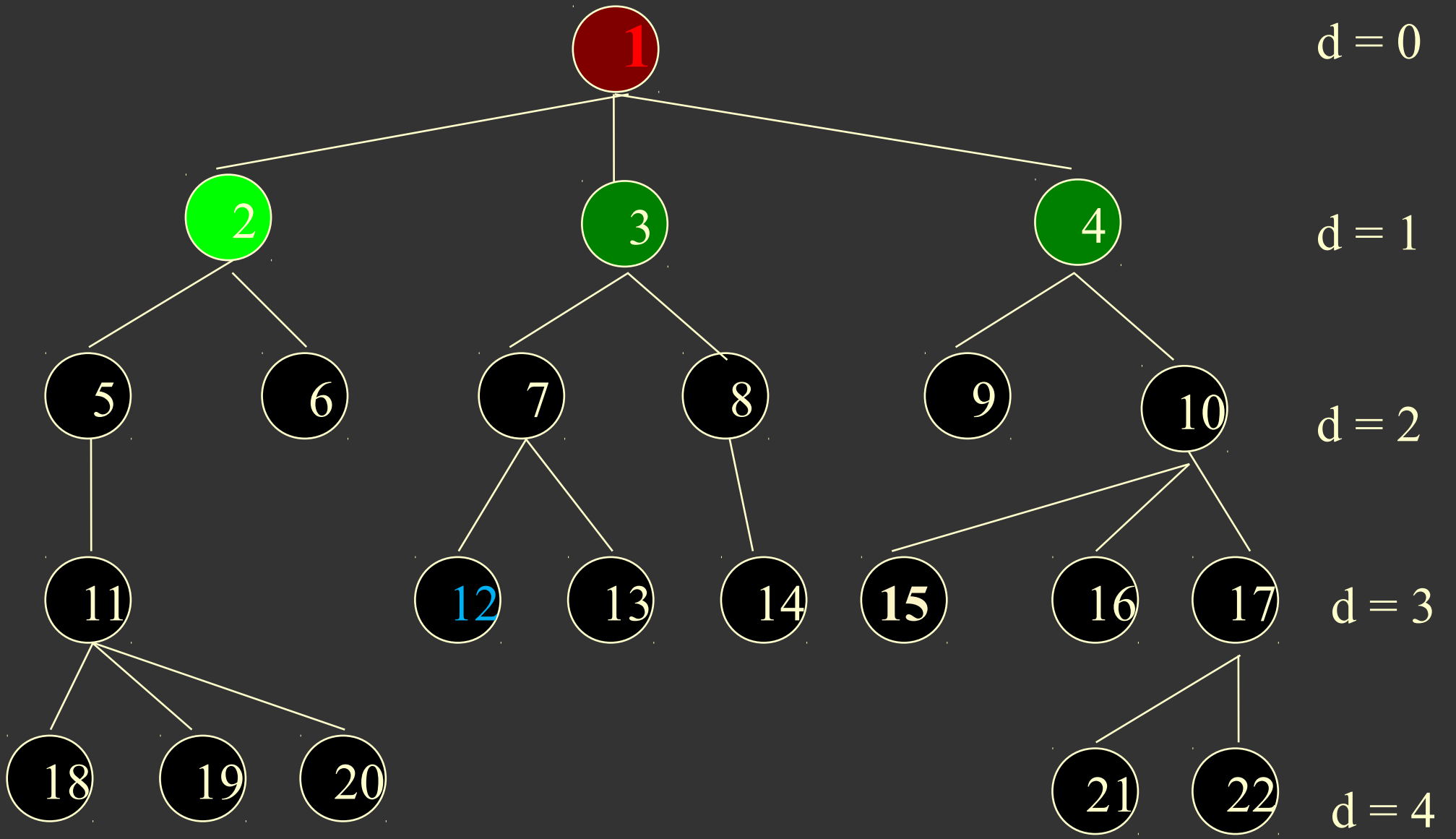
3.2.2 Adicionar ao final de F todos os sucessores de n que não estejam em F ou E

3.2.3 Voltar ao passo 2



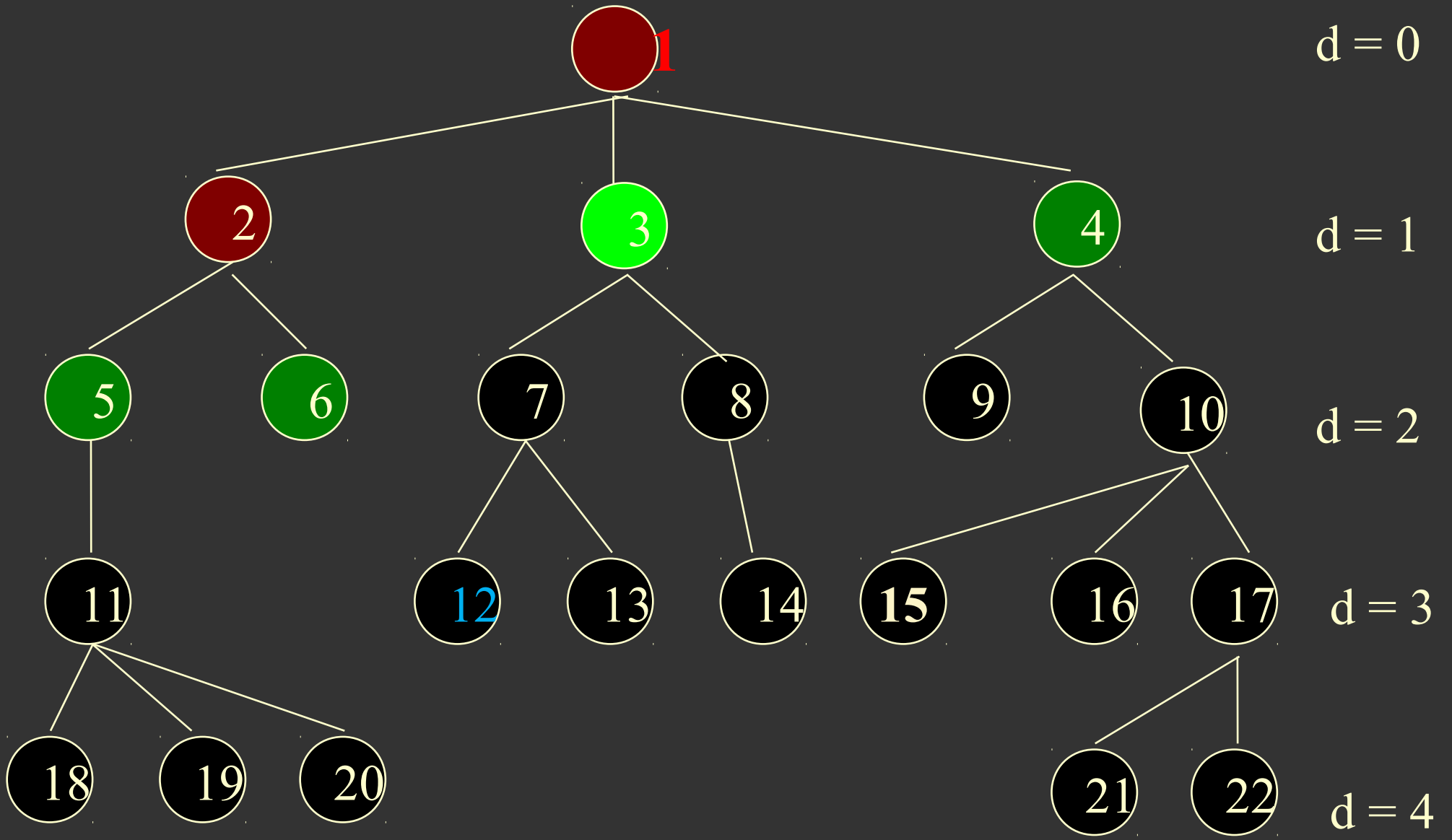
F = {1}

E = {}



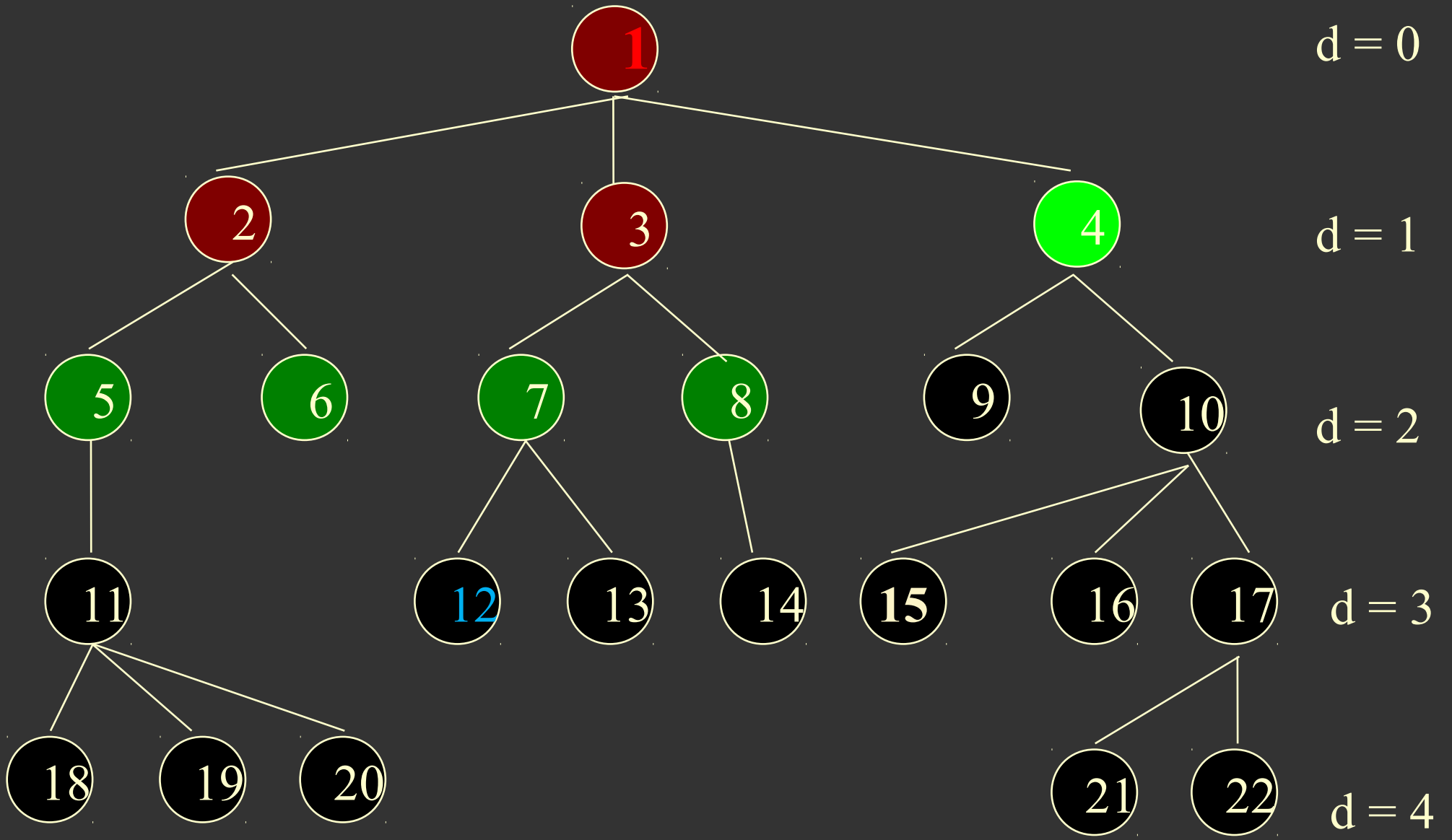
F = {2,3,4}

E = {1}



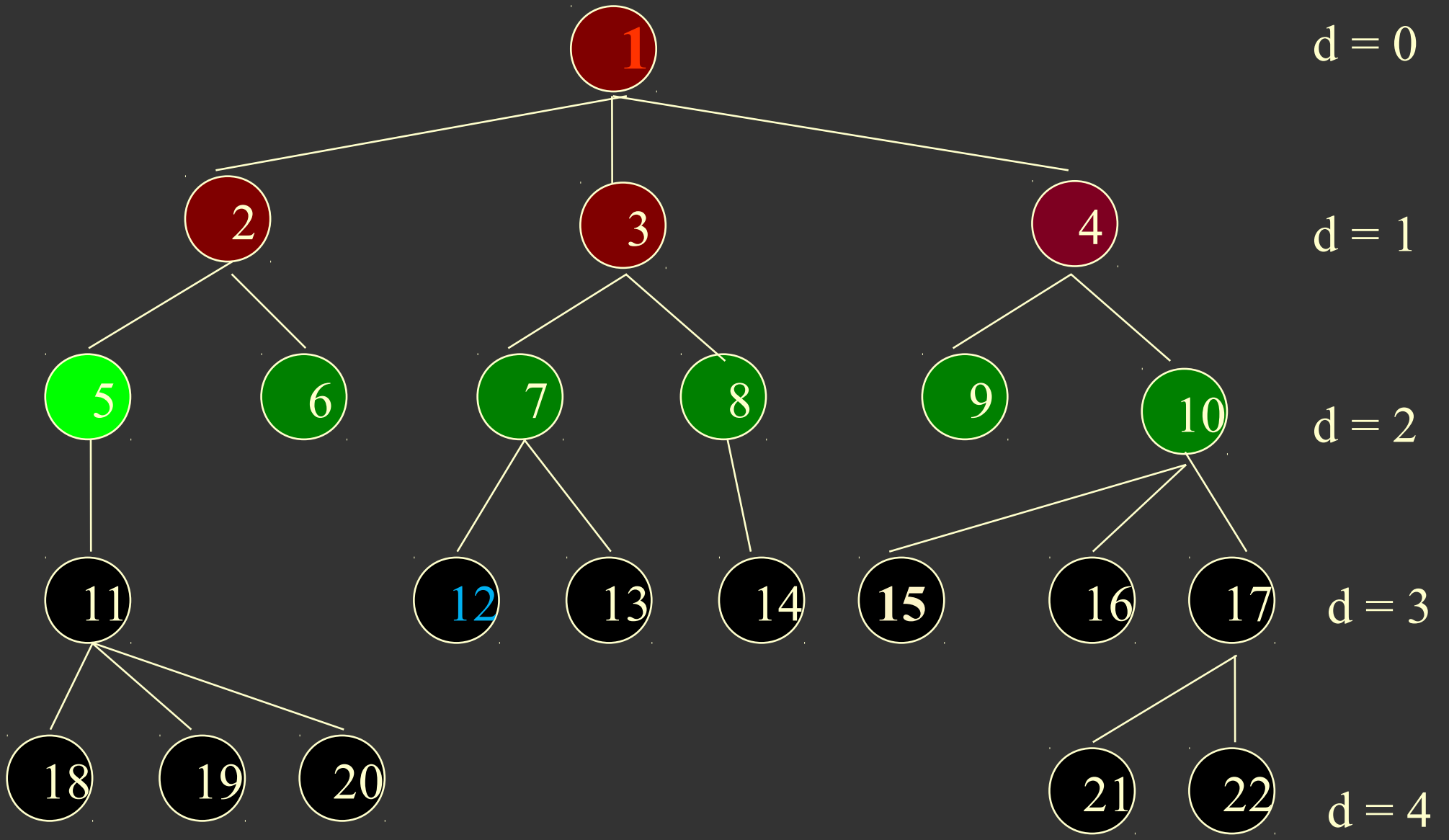
F = {3,4,5,6}

E = {1,2}



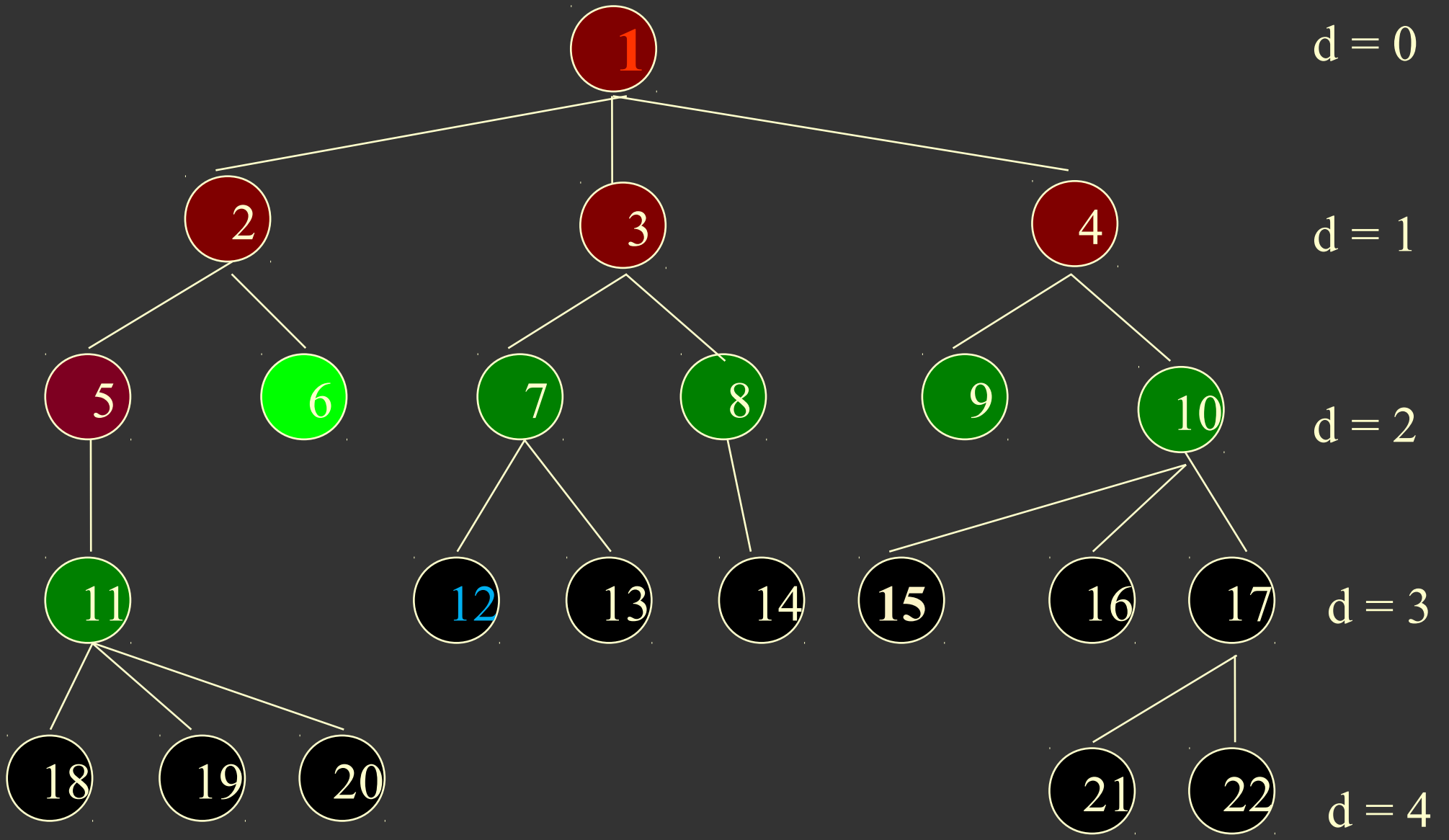
F = {4,5,6,7,8}

E = {1,2,3}



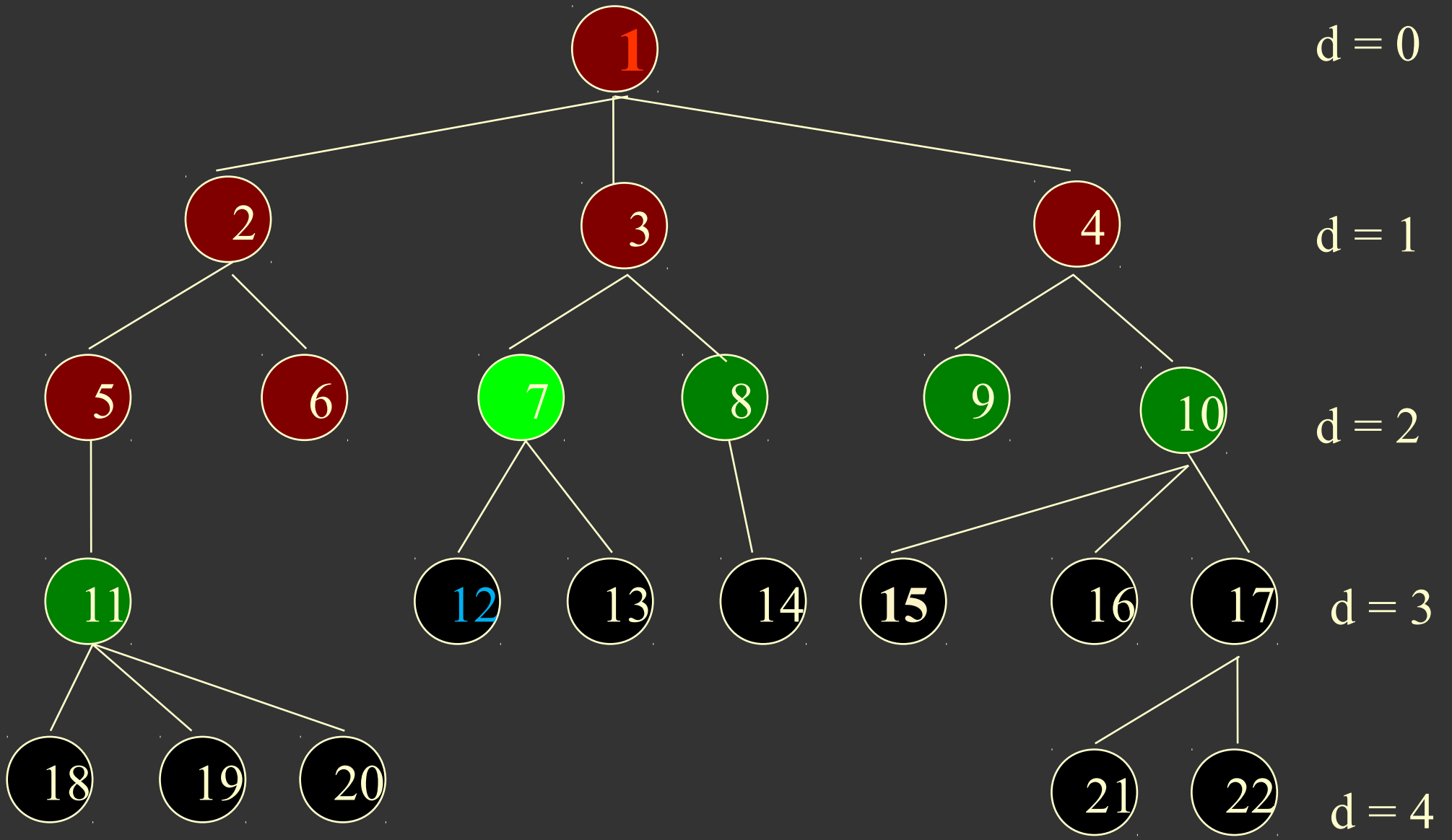
F = {5,6,7,8,9,10}

E = {1,2,3,4}



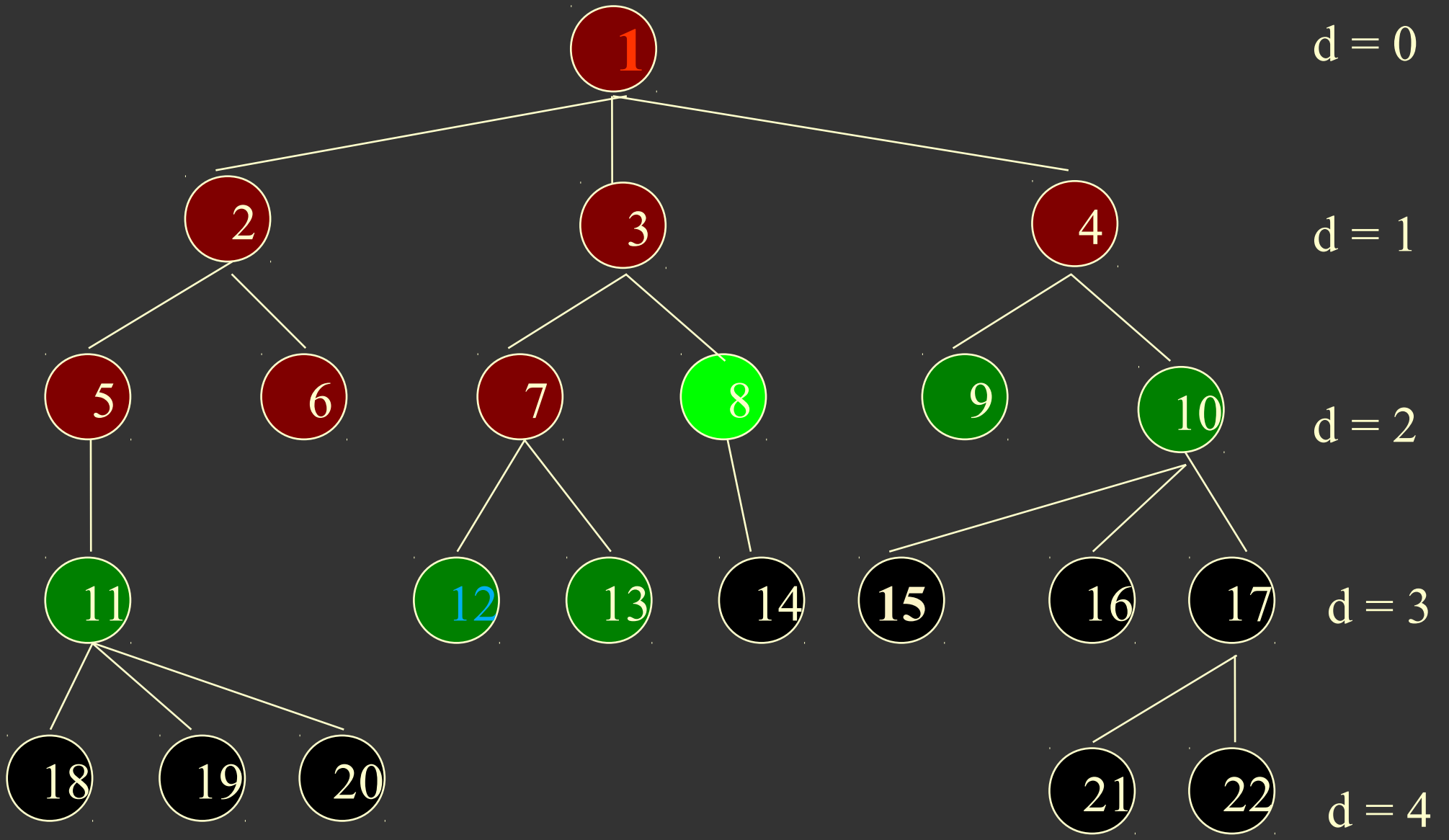
F = {6,7,8,9,10,11}

E = {1,2,3,4,5}



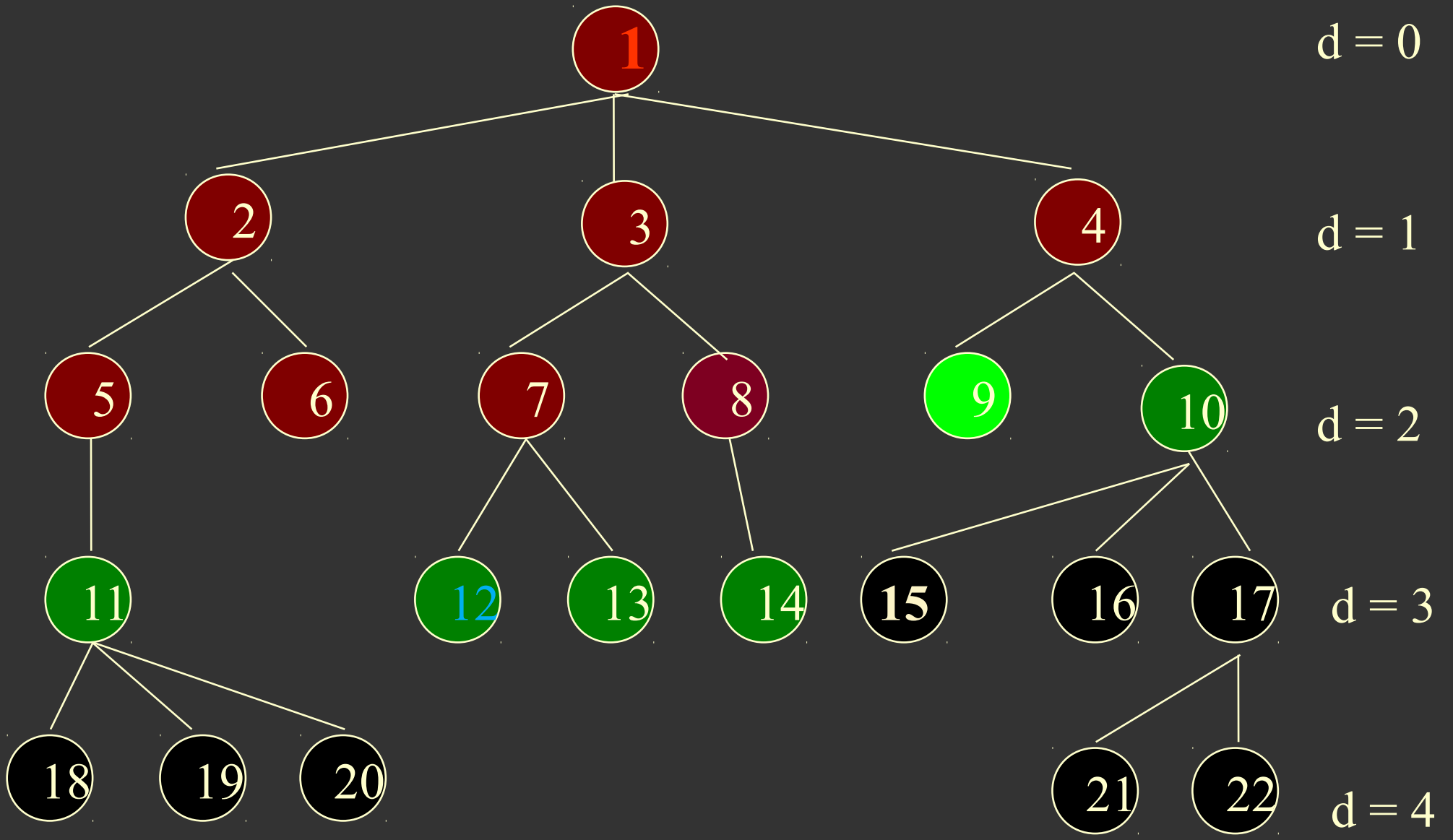
F = {7,8,9,10,11}

E = {1,2,3,4,5,6}



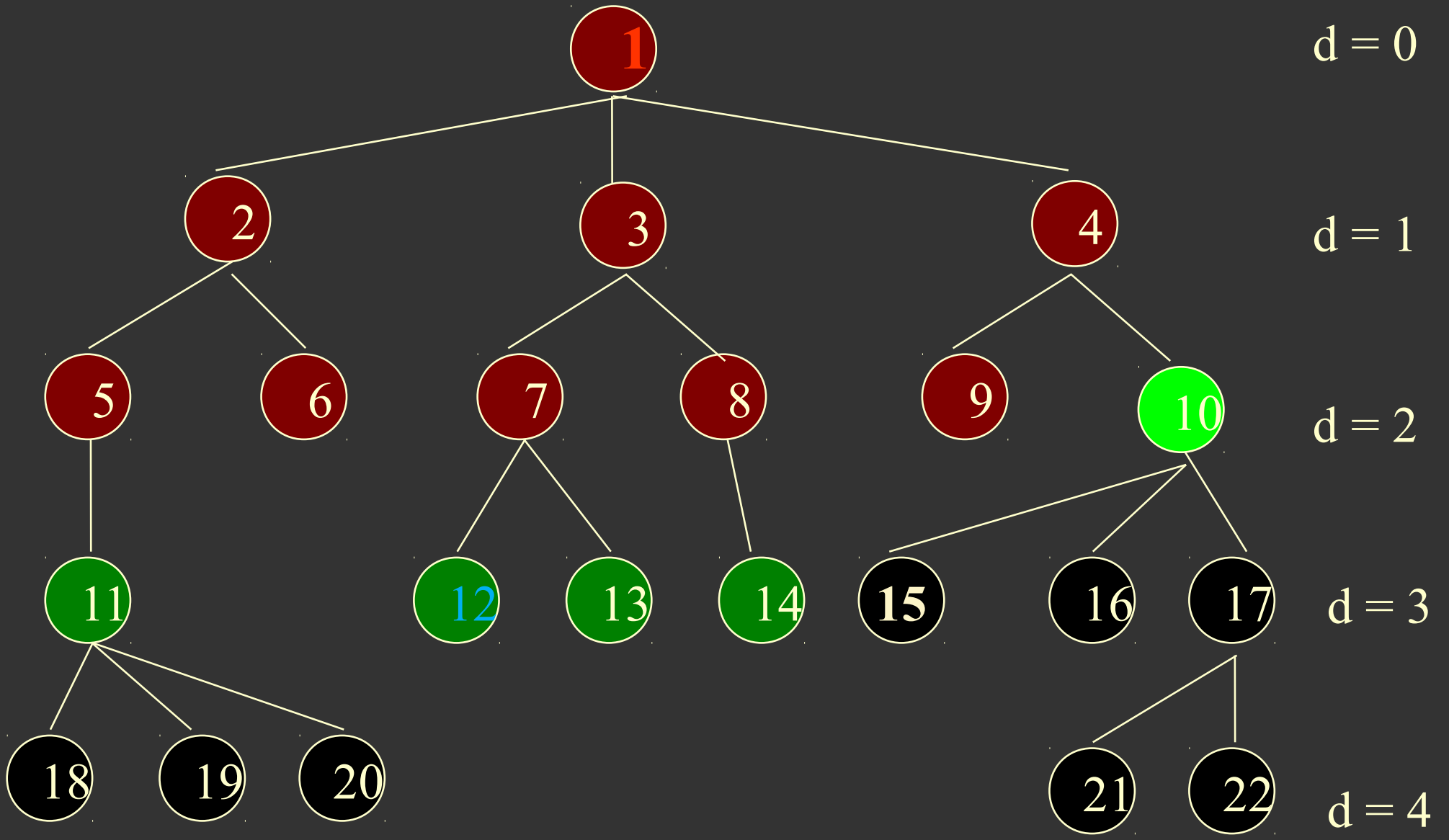
F = {8,9,10,11,12,13}

E = {1,2,3,4,5,6,7}



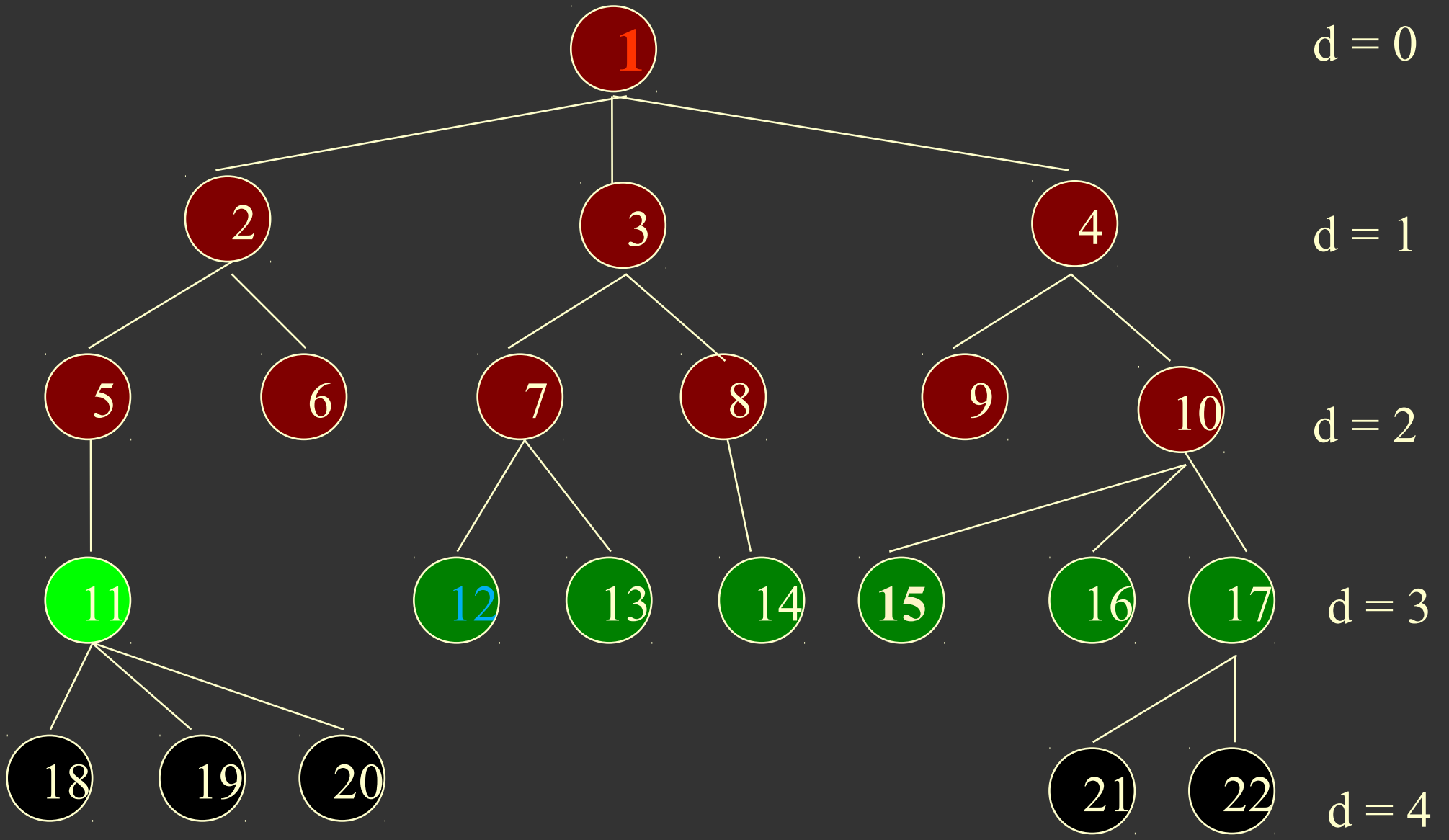
F = {9,10,11,12,13,14}

E = {1,2,3,4,5,6,7,8}



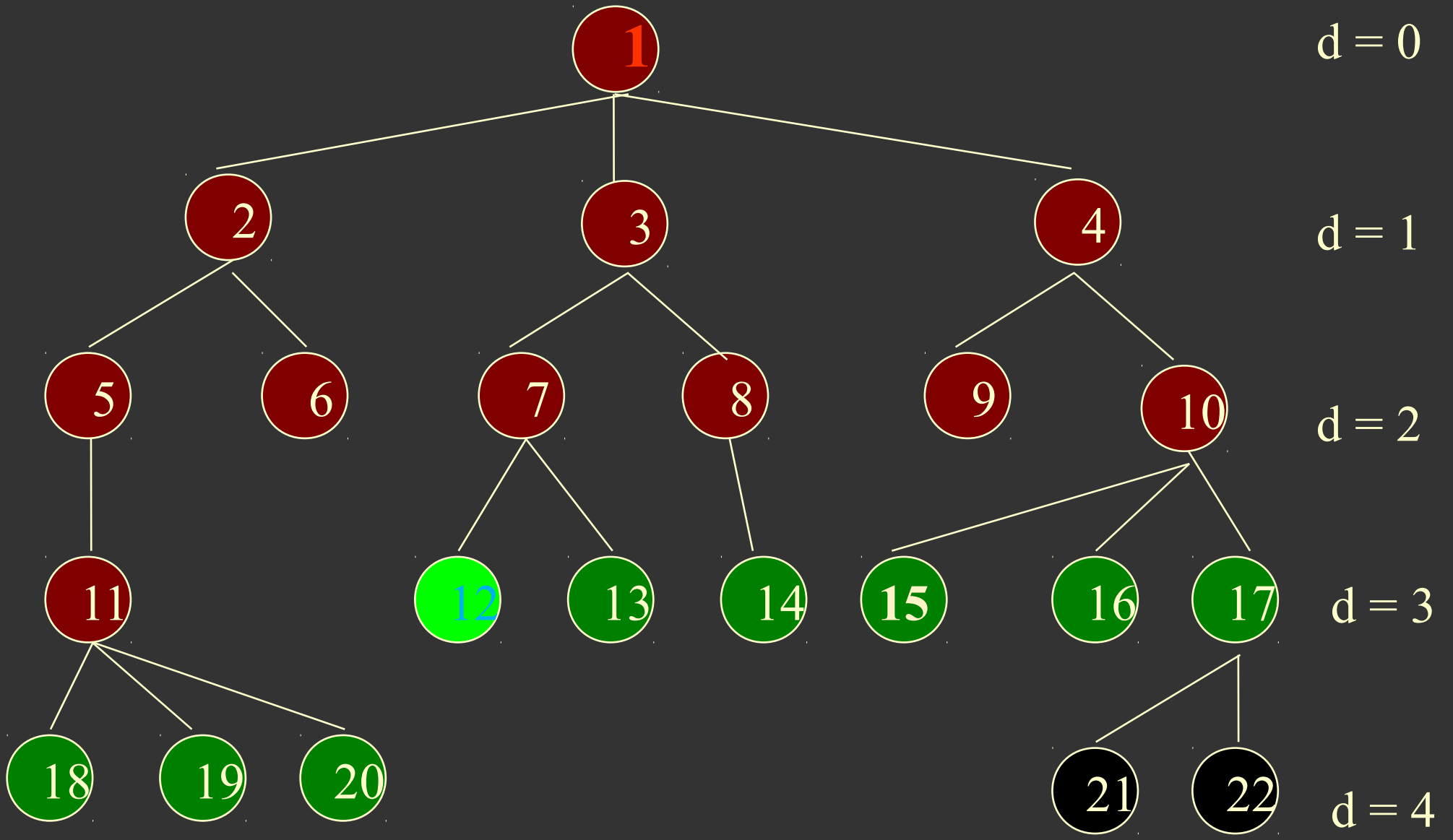
F = {10,11,12,13,14}

E = {1,2,3,4,5,6,7,8,9}



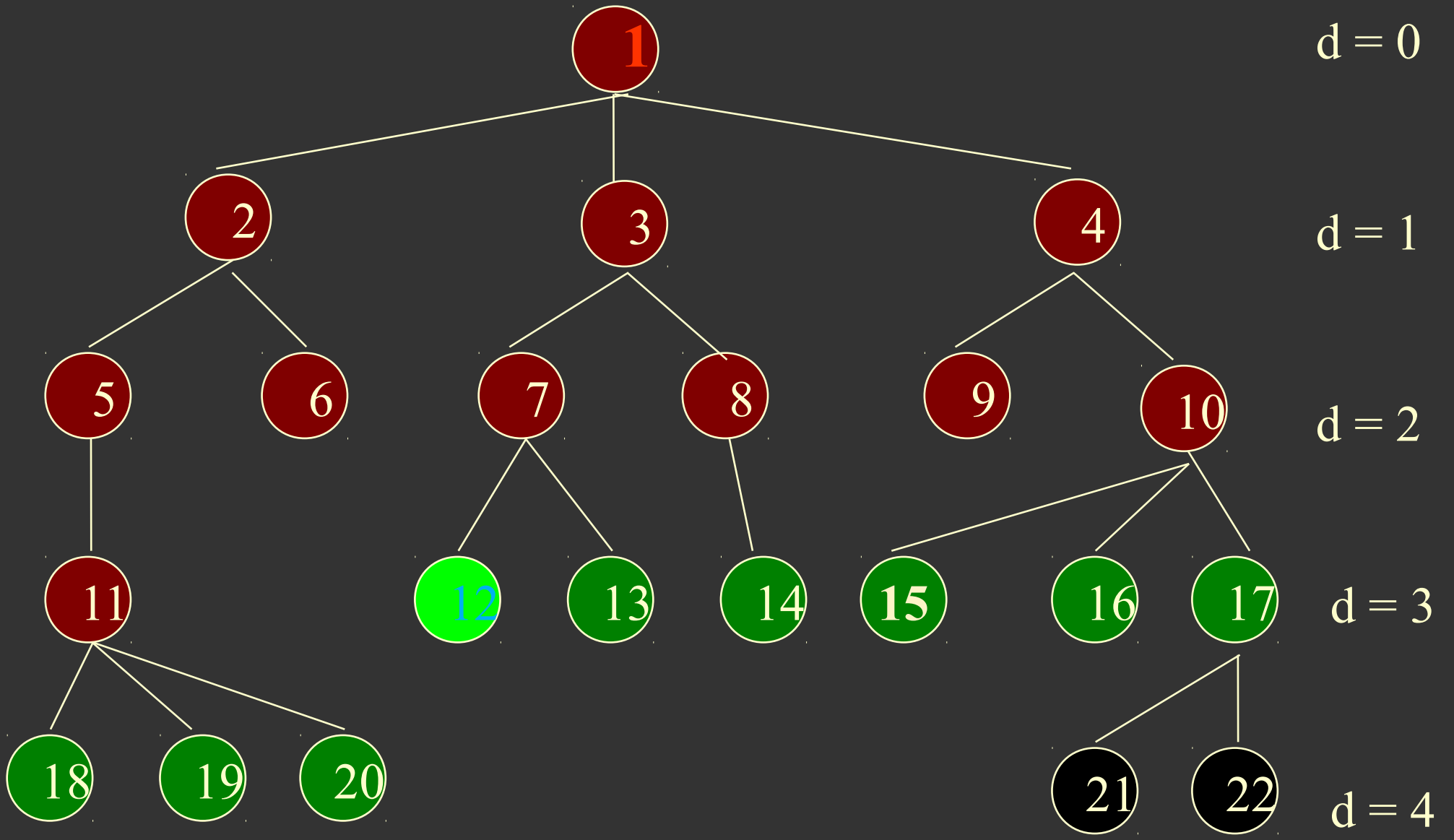
F = {11,12,13,14,15,16,17}

E = {1,2,3,4,5,6,7,8,9,10}



F = {12,13,14,15,16,17,18,19,20}

E = {1,2,3,4,5,6,7,8,9,10,11}



- Meta encontrada, mas cadê o caminho?

Caminho

- Na maioria das aplicações, não basta encontrar o objetivo
  - É preciso encontrar o caminho entre o estado inicial e o estado objetivo
  - Em alguns casos, é preciso armazenar as ações também!
- Para isso, armazenamos listas com os rótulos dos estados explorados que compõem o caminho
- Exemplo:
  - [12,7,3,1] % lista que indica o caminho do no inicial ao no meta

Algoritmo BL

1 Inserir os nós iniciais na lista de busca F % fronteira

2 Se F é vazio

2.1 Então a busca não foi bem sucedida

3 Senão seja n o primeiro estado de F

3.1 Se n é um estado meta então

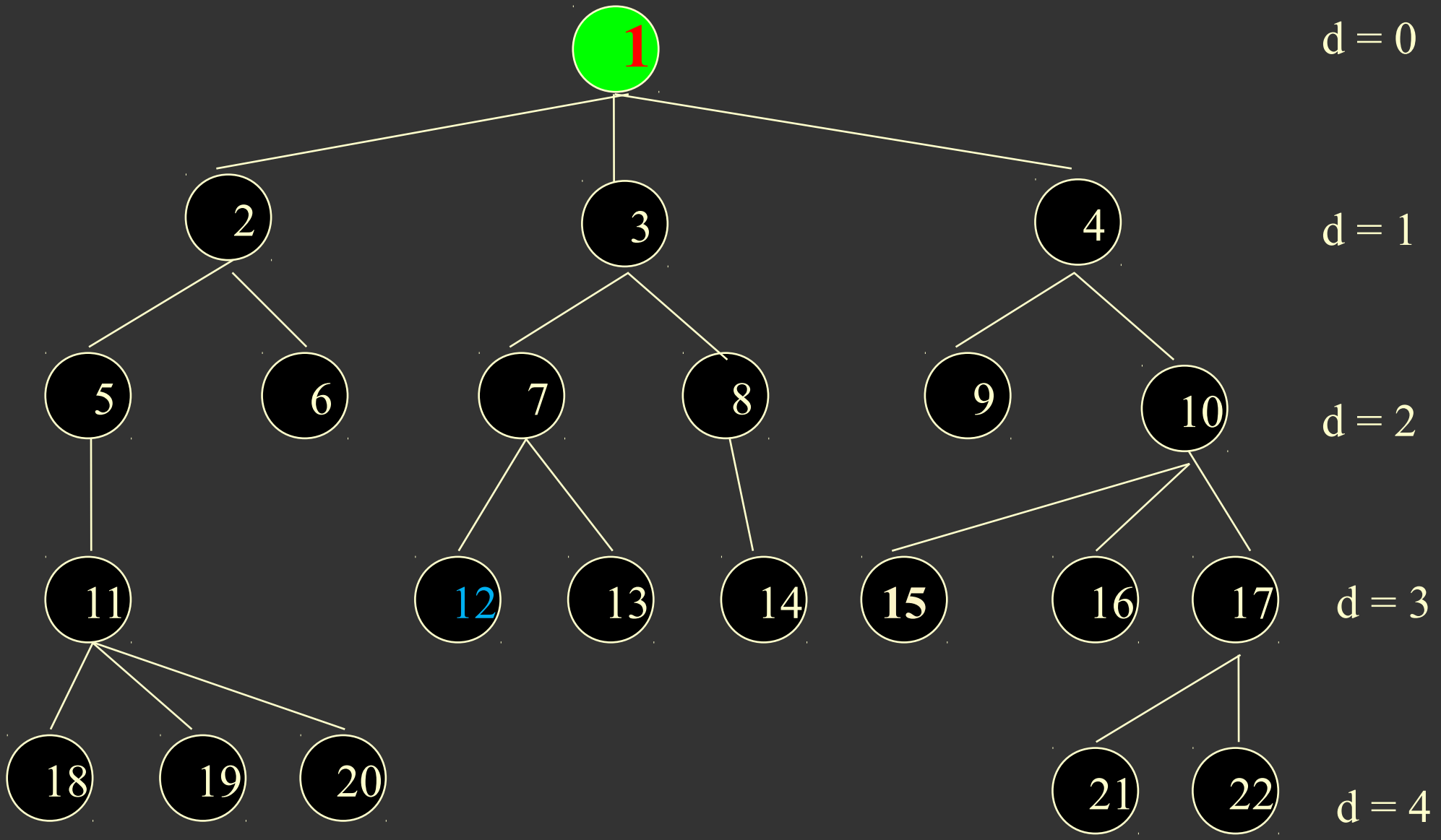
3.1.1 Retornar n

3.2 Senão

3.2.1 Remover n de F e inserir em E % explorados

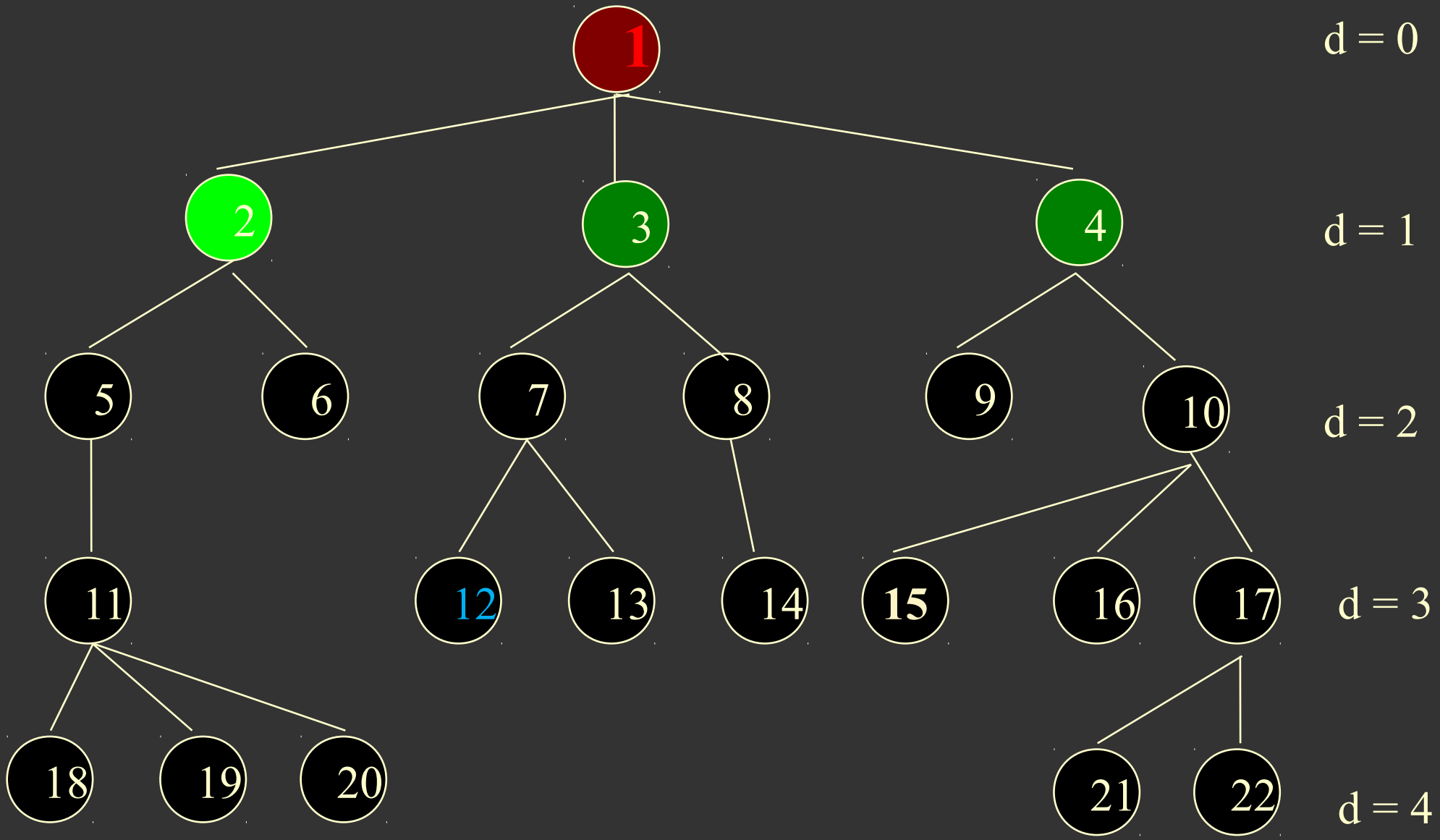
3.2.2 Adicionar ao final de F todos os sucessores de n que não estejam em F ou E junto com n e o caminho até o estado inicial

3.2.3 Voltar ao passo 2



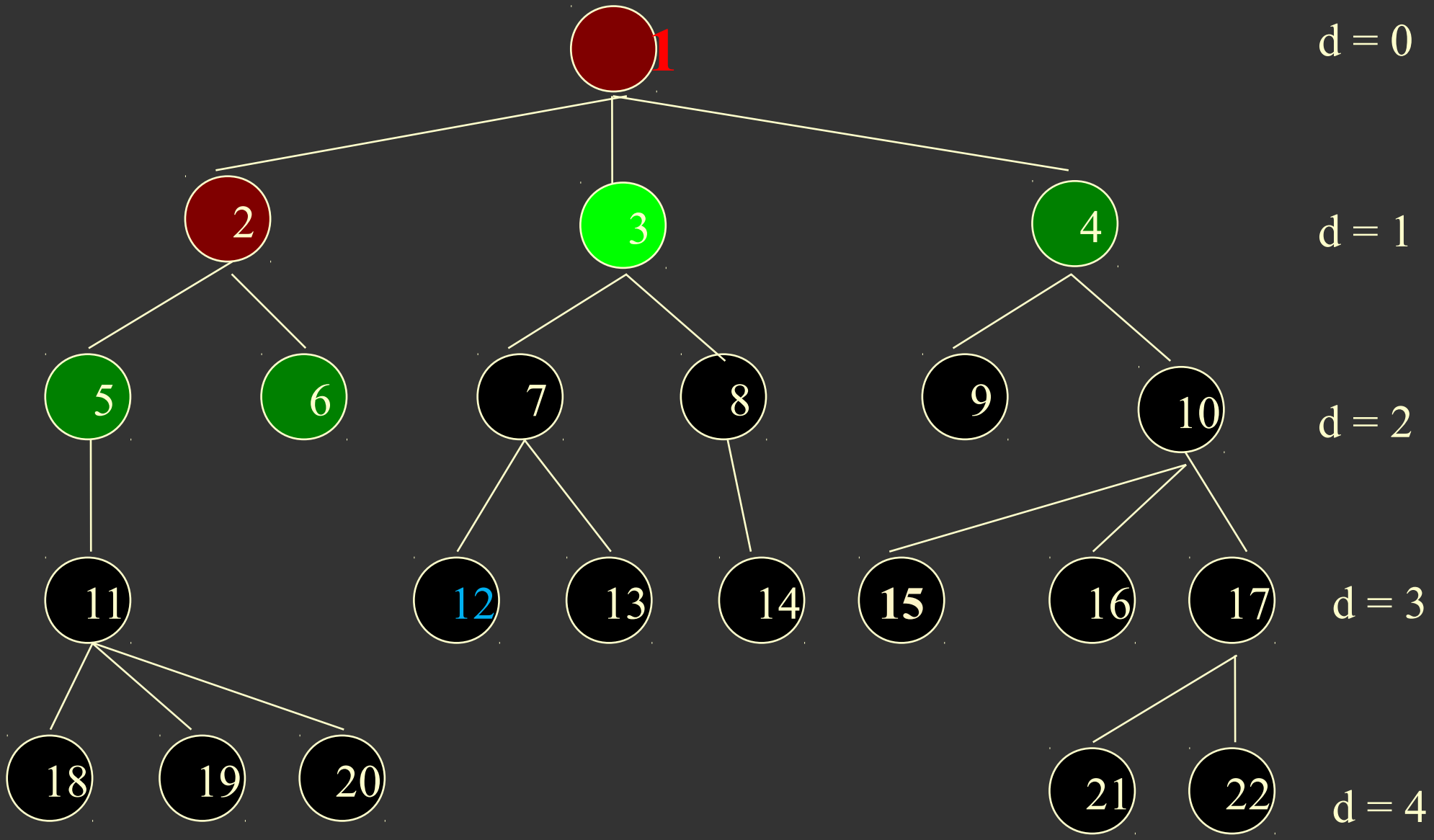
F = {[1]}

E = {}



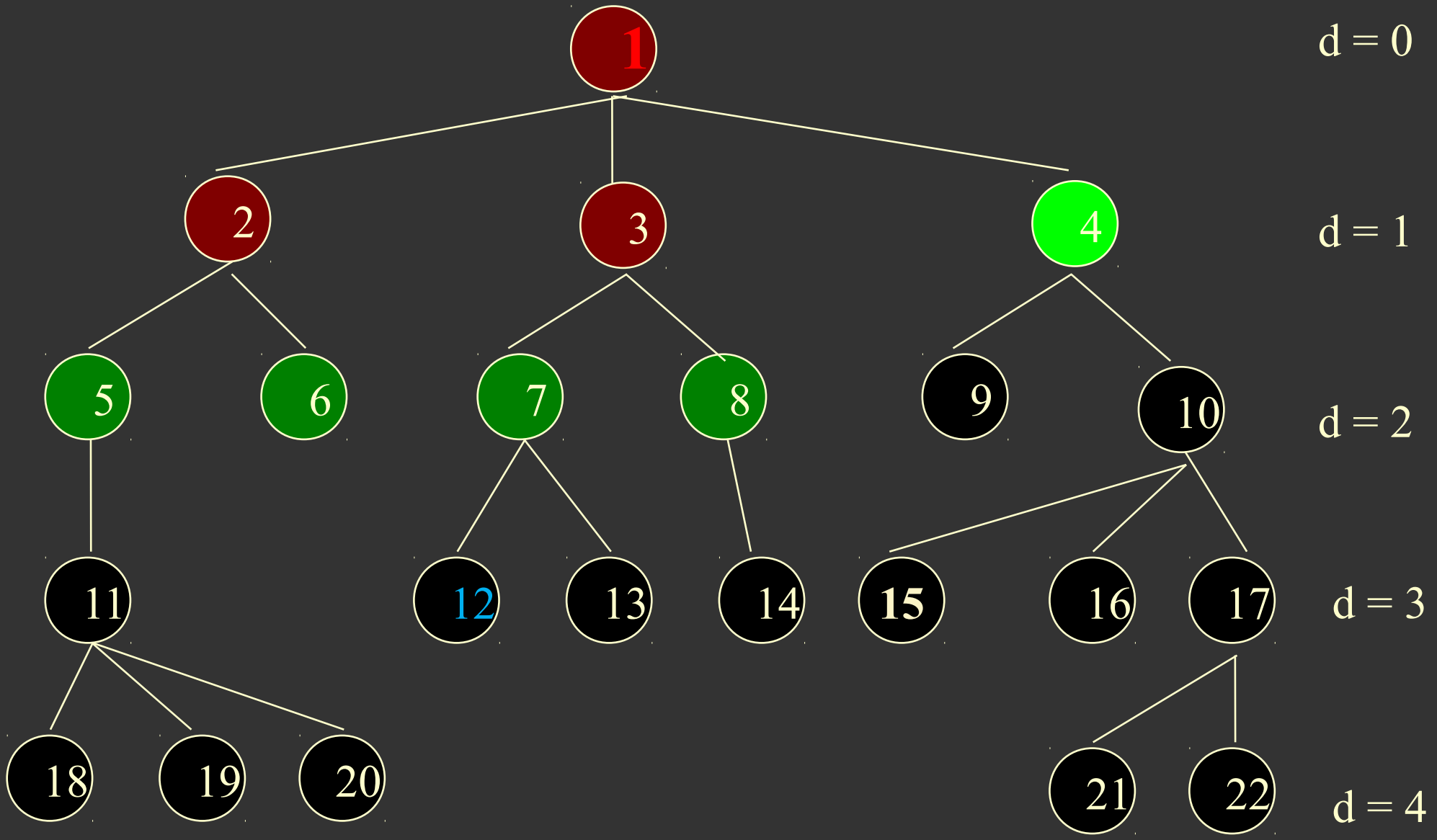
F = {[2,1],[3,1],[4,1]}

E = {1}



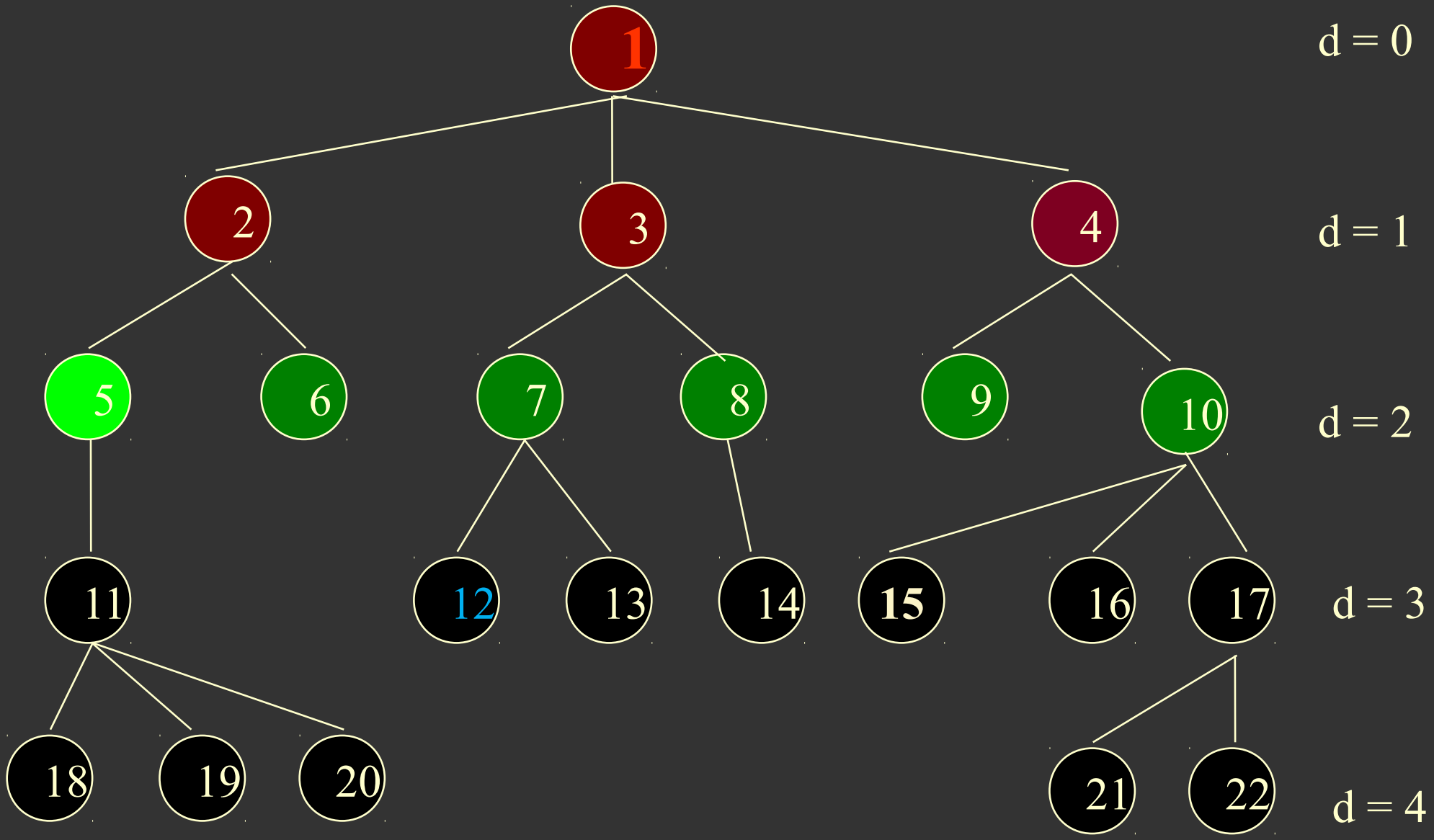
F = {[3,1],[4,1],[5,2,1],[6,2,1]}

E = {1,2}



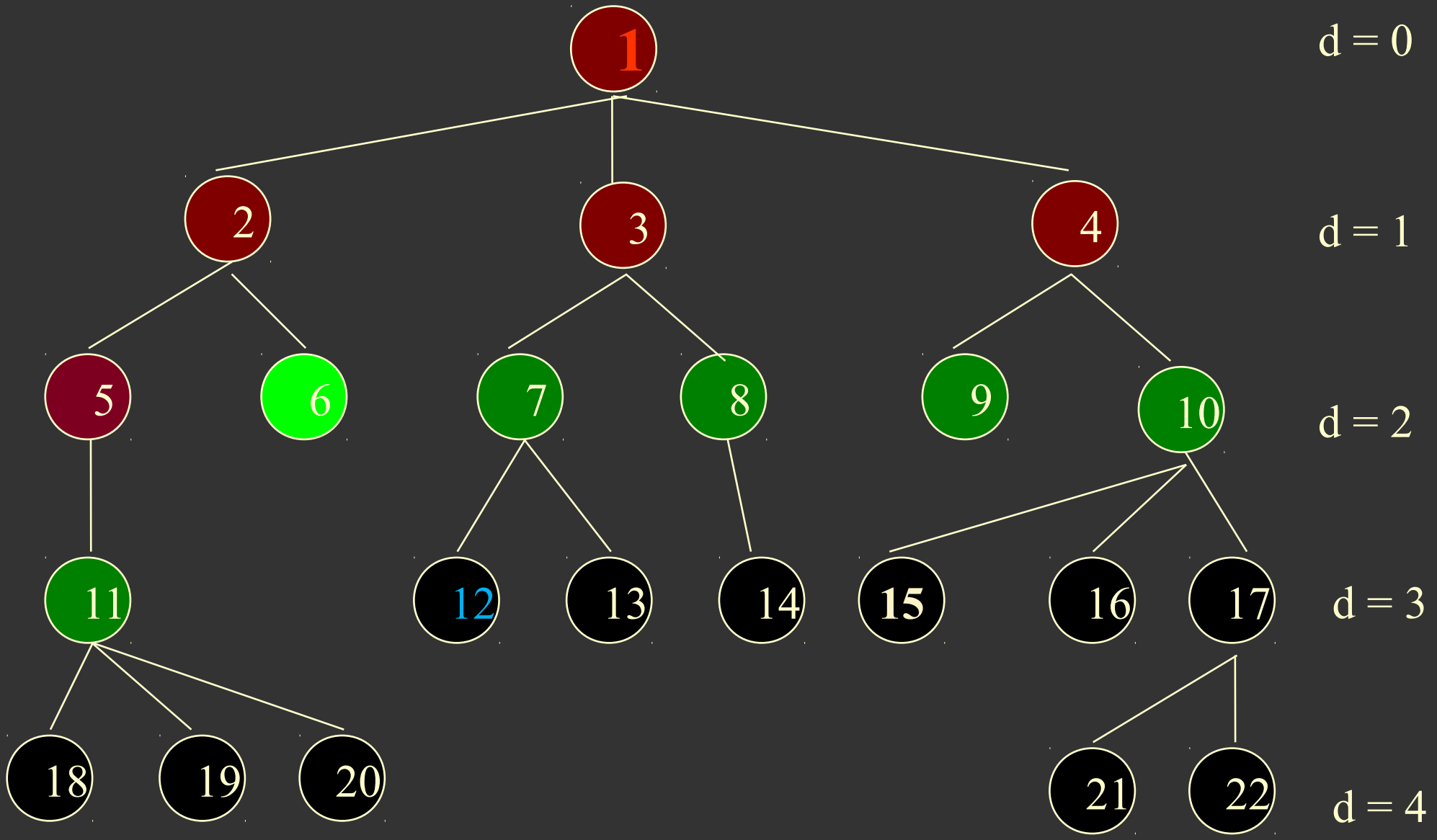
F = {[4,1],[5,2,1],[6,2,1],[7,3,1],[8,3,1]}

E = {1,2,3}



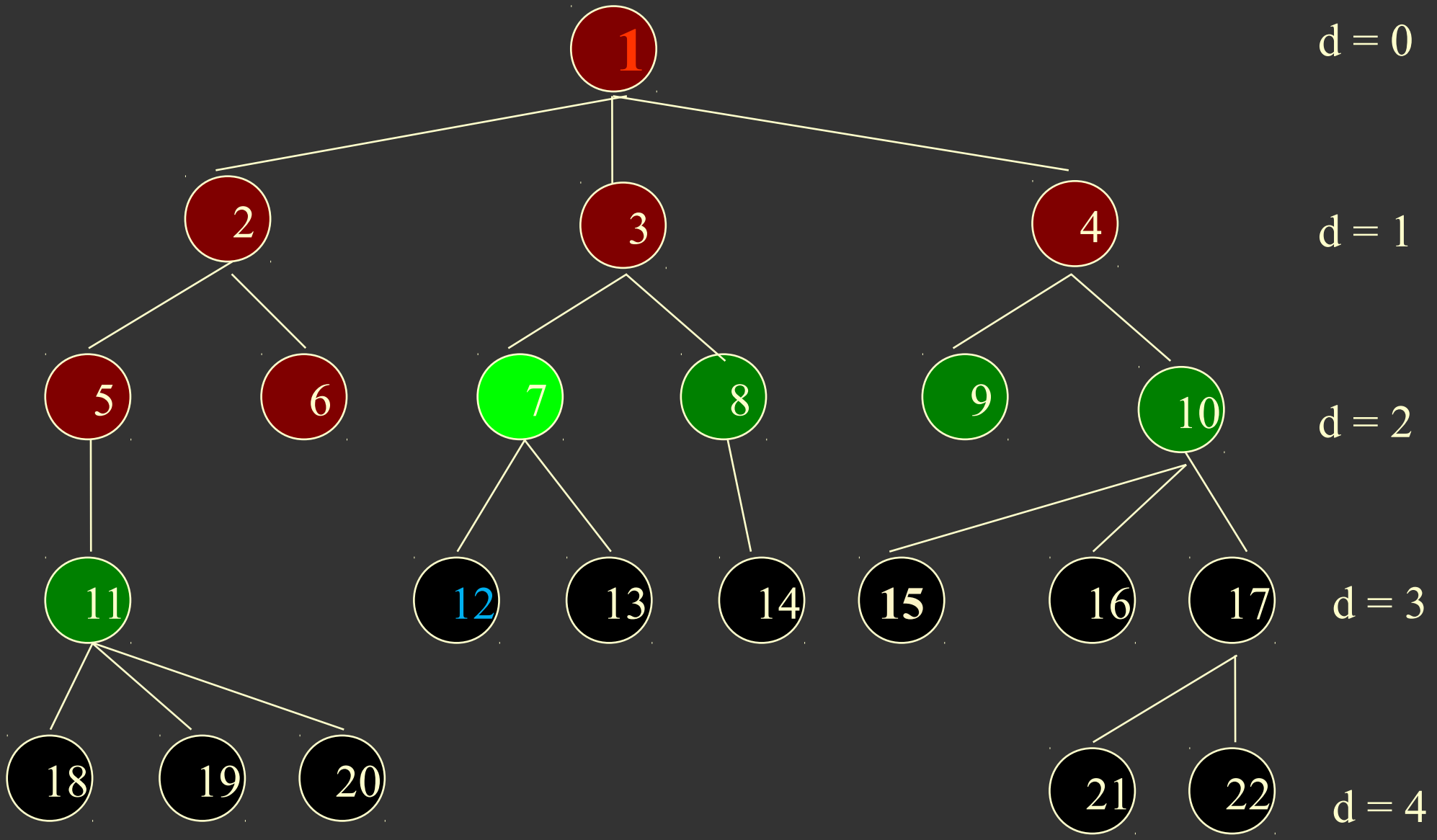
F = {[5,2,1],[6,2,1],[7,3,1],[8,3,1],[9,4,1],[10,4,1]}

E = {1,2,3,4}



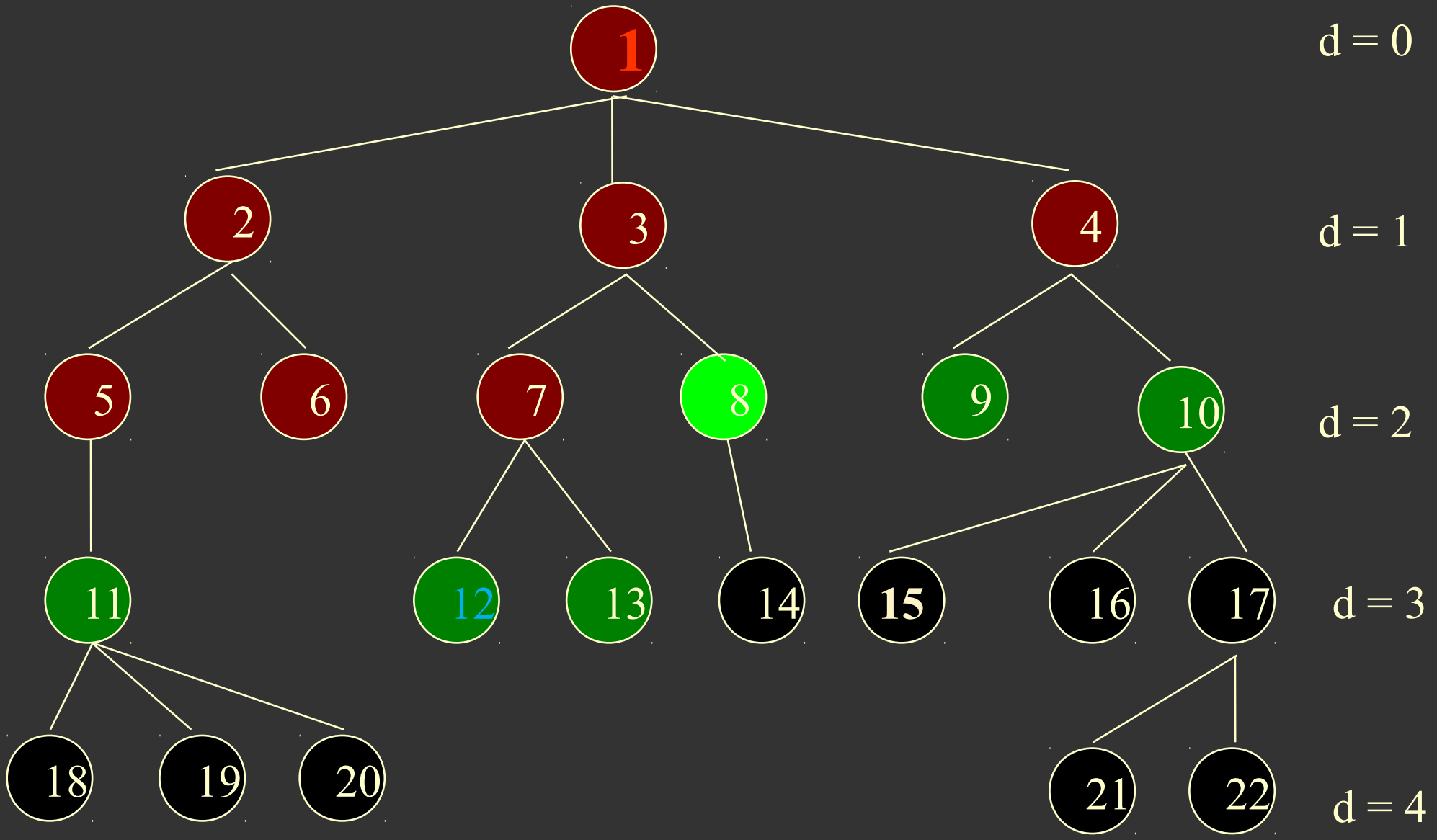
F = {[6,2,1],[7,3,1],[8,3,1],[9,4,1],[10,4,1],
[11,5,2,1]}

E = {1,2,3,4,5}



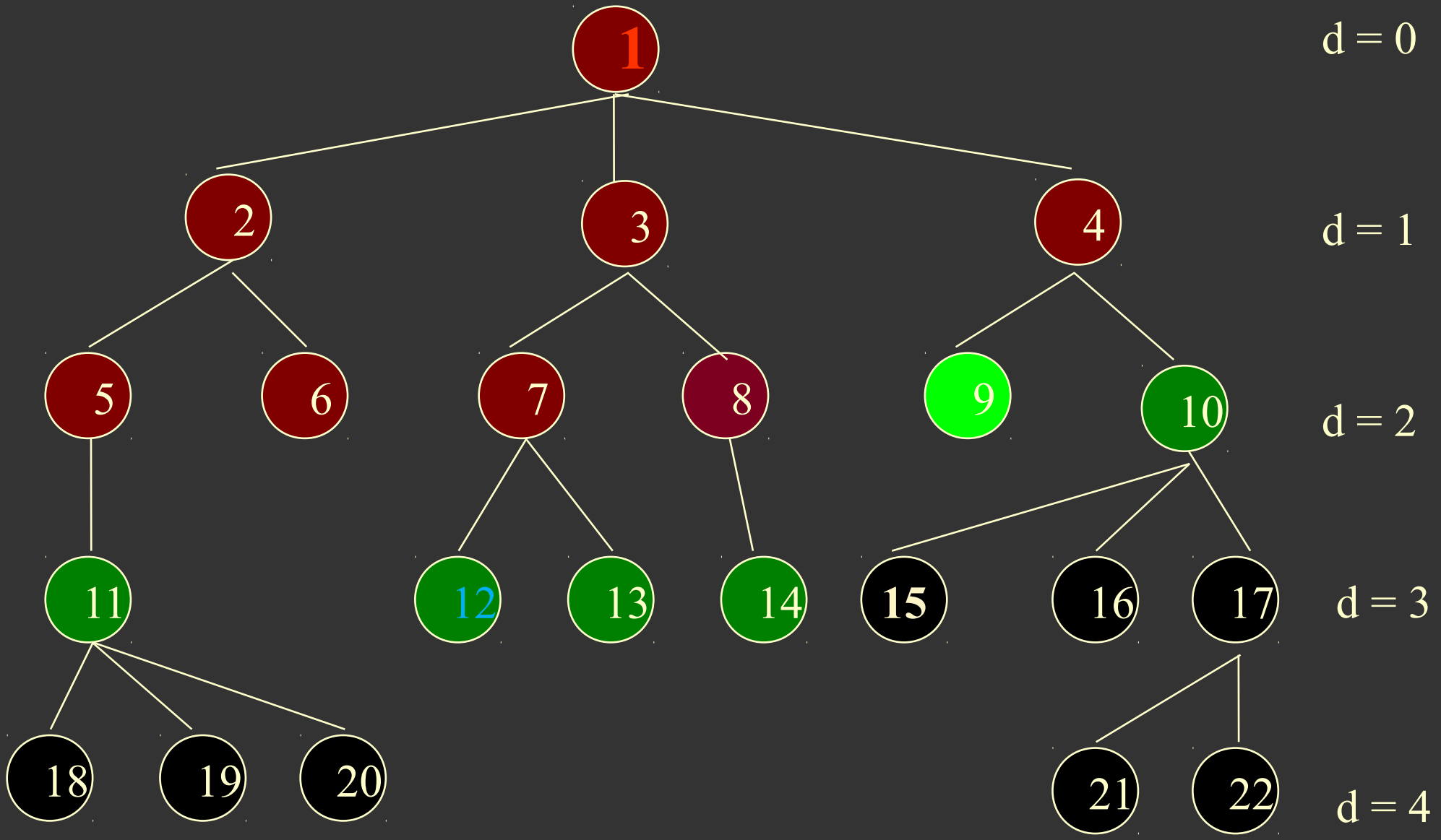
F = {[7,3,1],[8,3,1],[9,4,1],[10,4,1],[11,5,2,1]}

E = {1,2,3,4,5,6}



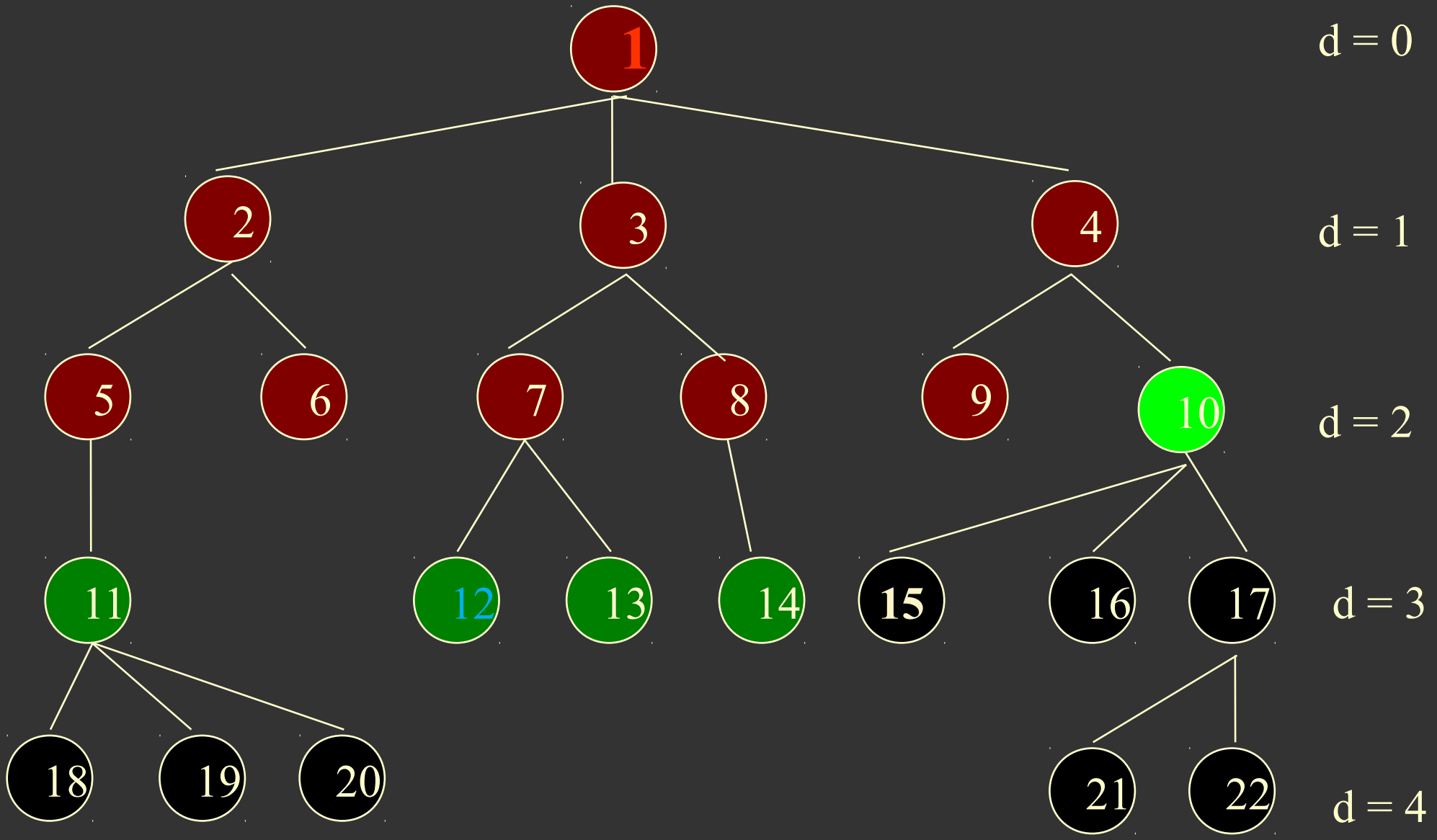
F = {[8,3,1],[9,4,1],[10,4,1],[11,5,2,1],[12,7,3,1],
[13,7,3,1]}

E = {1,2,3,4,5,6,7}



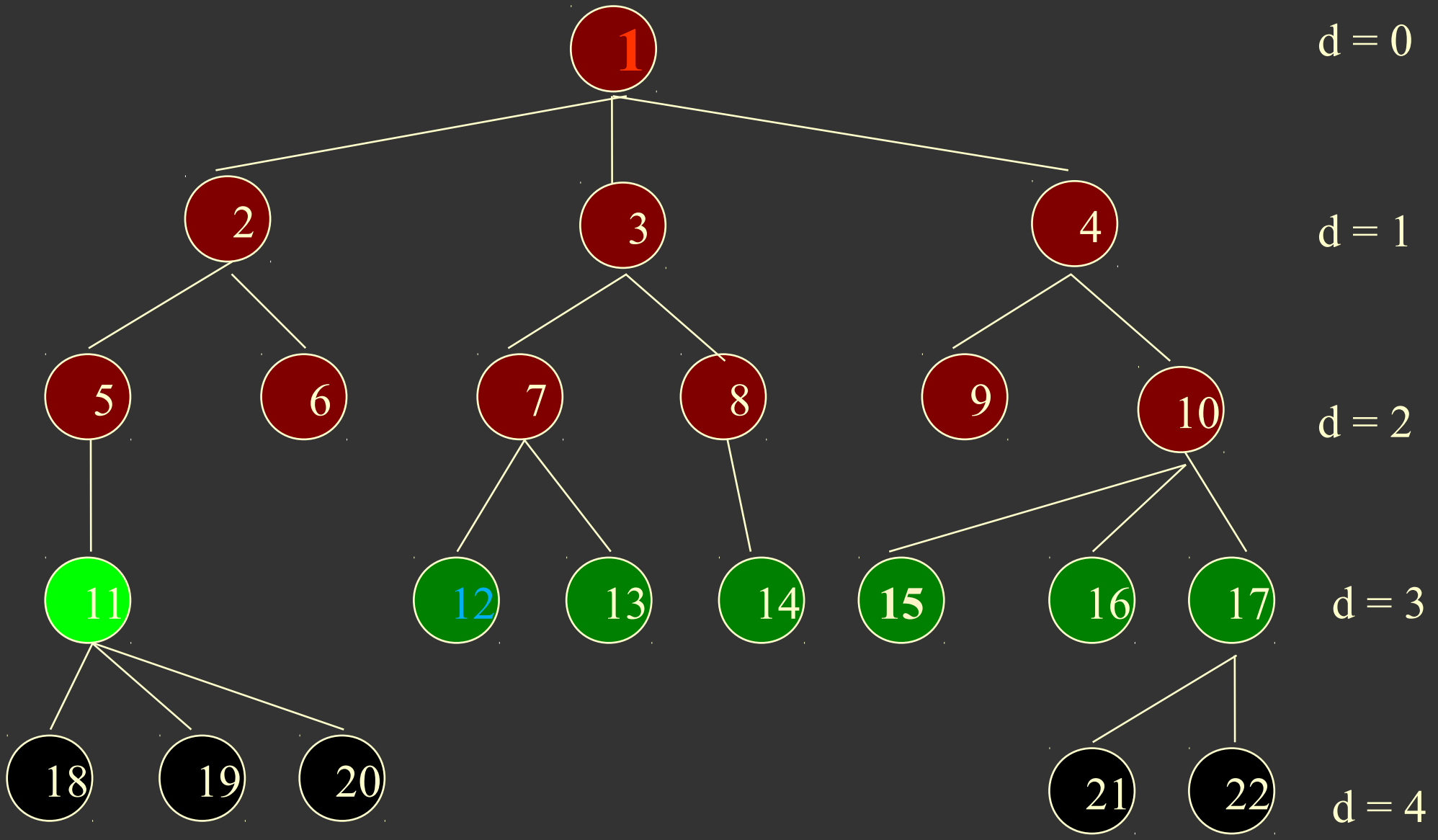
F = {[9,4,1],[10,4,1],[11,5,2,1],[12,7,3,1],
[13,7,3,1],[14,8,3,1]}

E = {1,2,3,4,5,6,7,8}



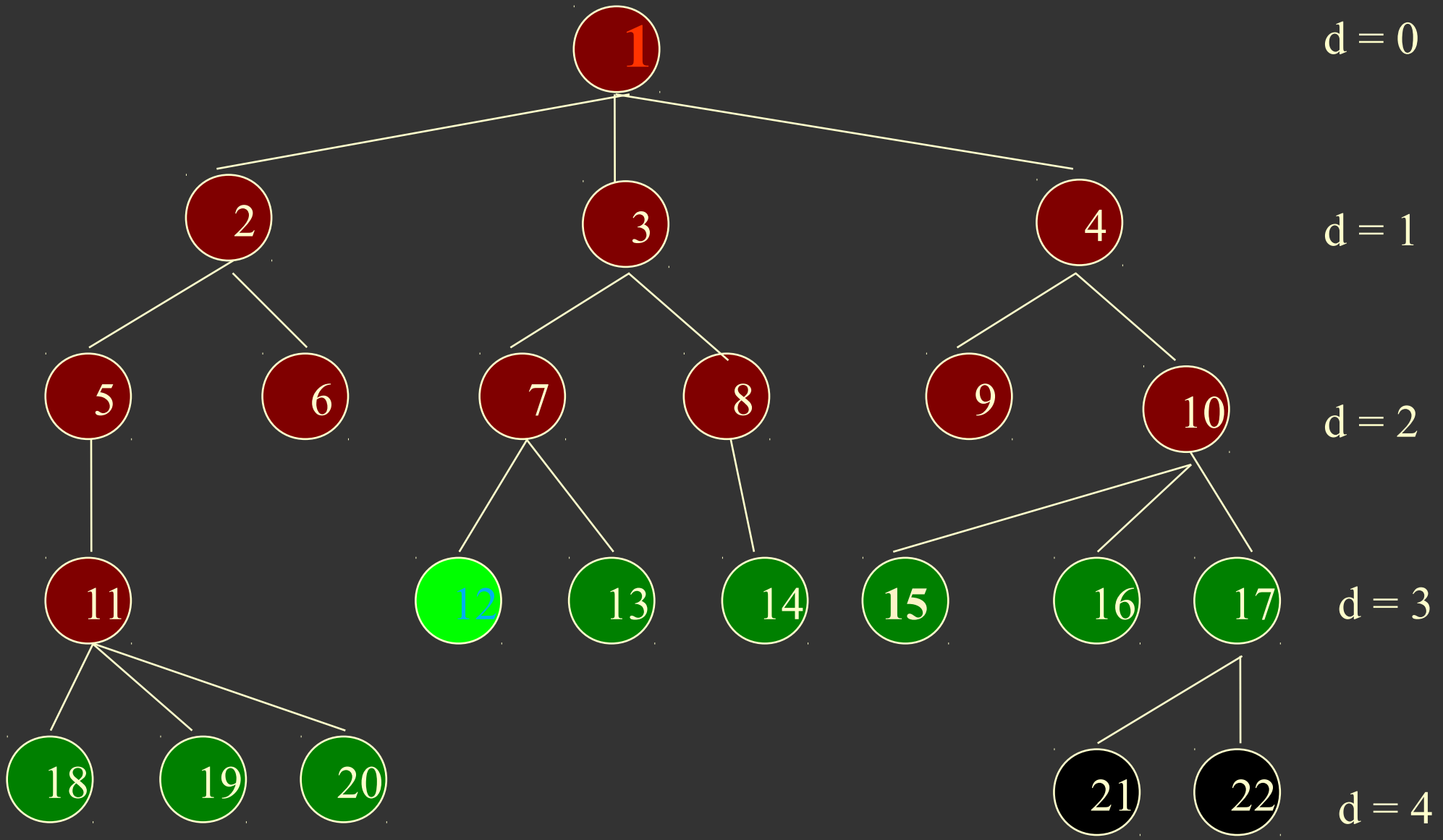
F = {[10,4,1],[11,5,2,1],[12,7,3,1],[13,7,3,1],
[14,8,3,1]}

E = {1,2,3,4,5,6,7,8,9}



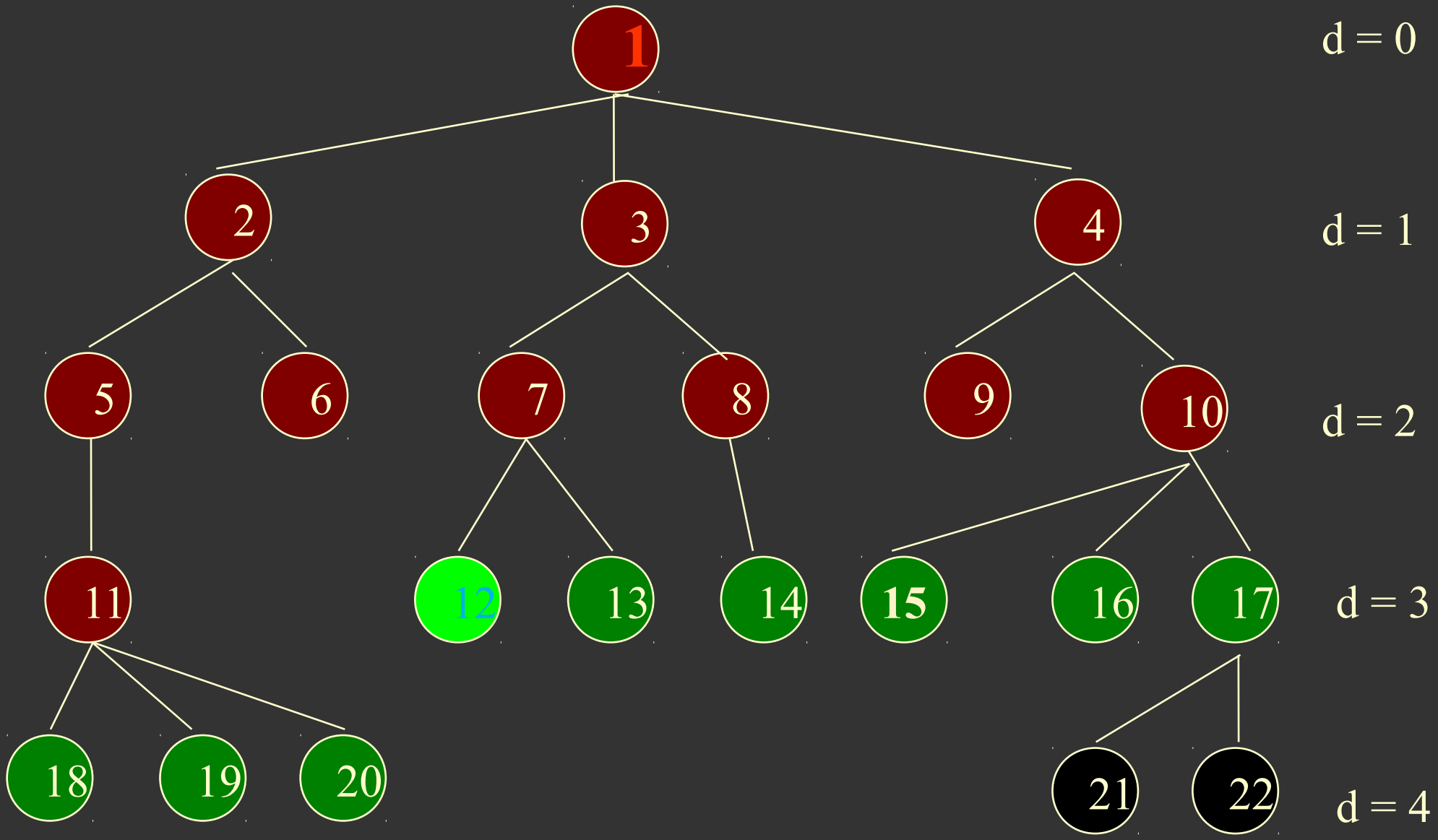
F = {[11,5,2,1],[12,7,3,1],[13,7,3,1],[14,8,3,1],
[15,10,4,1],[16,10,4,1],[17,10,4,1]}

E = {1,2,3,4,5,6,7,8,9,10}



F = {[12,7,3,1],[13,7,3,1],[14,8,3,1],[15,10,4,1],[16,10,4,1],
[17,10,4,1],[18,11,5,2,1],[19,11,5,2,1],[20,11,5,2,1]}

E = {1,2,3,4,5,6,7,8,9,10,11}



- [12,7,3,1] % meta encontada = cabeça

  % cauda é o caminho

Propriedades BL

- Seja:
  - b – número máximo (pior caso) de filhos de um nó
  - d - profundidade da solução de custo mínimo
  - m - profundidade máxima do espaço de estados

Propriedades BL

- Completo?? Sim, se b é finito.
- Tempo?? $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- Espaço?? $O(b^d)$ (mantém todos os nós expandidos na memória no pior caso)
- Admissível?? Sim, sempre encontra o menor caminho.
- Como mostrado, espaço pode ser grande problema, especialmente se d for grande.

Comando Bagof

- Utilizar a função do Prolog bagof
  - bagof(X,P,L)
- Ela produz uma lista L com todos os objetos que possam ser unificados com X e que satisfazem P
  - Exemplo:
    - idade(joao,5)
    - idade(ana,4)
    - idade(maria,5)
    - ?- bagof(X,idade(X,5),L)
    - L = [joao,maria]

Código Prolog BL

- Começamos encapsulamento da busca

```
%solucao por busca em largura (bl)

solucao_bl(Inicial,Solucao) :- bl([[Inicial]],Solucao).
```

- que recebe o estado inicial e retorna estado final (solução ou meta)
- Procura a meta no elemento da cabeça da lista de fronteira, descartando o resto

```
%Se o primeiro estado de F for meta, então o retorna com o caminho

bl([[Estado|Caminho]|_],[Estado|Caminho]) :- meta(Estado).
```

Código Prolog BL

- Se a meta não foi encontrada no “Primeiro” da fronteira (regra do slide anterior), então precisamos procurar nos sucessores

```
%falha ao encontrar a meta, então estende o primeiro estado até seus sucessores e os coloca no final da lista de fronteira

bl([Primeiro|Outros], Solucao) :- estende(Primeiro,Sucessores), concatena(Outros,Sucessores,NovaFronteira), bl(NovaFronteira,Solucao).
```

- Os sucessores e seus caminhos são colocados em “Sucessores”(próximo slide), que é adicionado aos “Outros” estados de fronteira formando uma “NovaFronteira”

Código Prolog BL

- Estender significa encontrar o uma lista com os estados sucessores do primeiro de F

```
%metodo que faz a extensao do caminho até os nós filhos do estado
estende([Estado|Caminho],ListaSucessores):- bagof(
    [Sucessor,Estado|Caminho],
    (s(Estado,Sucessor),not(pertence(Sucessor,[Estado|Caminho]))),
    ListaSucessores),!.
%se o estado não tiver sucessor, falha e não procura mais (corte)
estende( _ ,[]).
```

- Não se pode aceitar caminhos com ciclos!
- Estados sem sucessores retornam [ ]

Alguns tipos de busca cega

- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Profundidade Iterativa

Busco de custo uniforme

- Busca admissível considerando custo
- Expande o nó de menor custo
- Insere nós em F em ordem ascendente do custo do caminho
- Os nós inseridos não podem estar em E, mas podem pertencer a fronteira F
- Exemplo aplicação:
  - Cálculo da melhor rota

Busca de custo uniforme

1 Inserir os nós iniciais na lista de busca F % fronteira

2 Se F é vazio

2.1 Então a busca não foi bem sucedida

3 Senão seja n o primeiro estado de F

3.1 Se n é um estado meta então

3.1.1 Retornar n

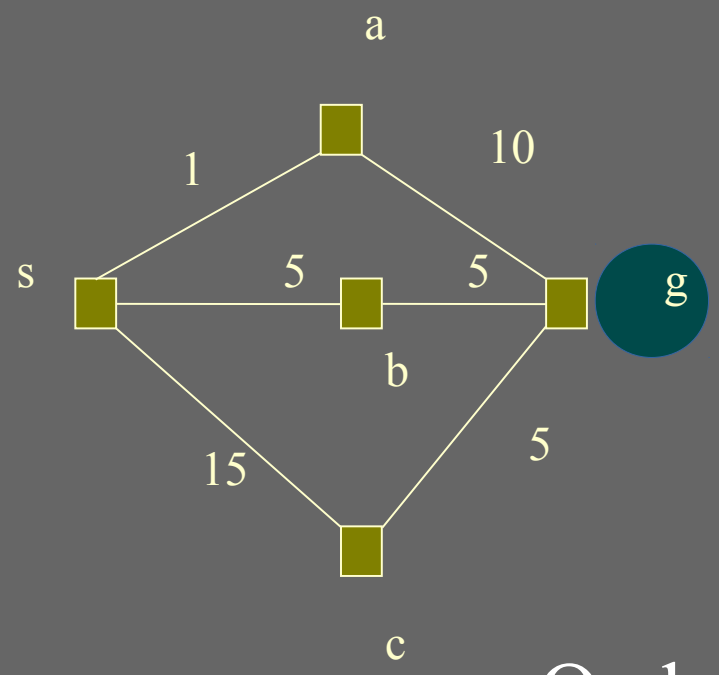
3.2 Senão

3.2.1 Remover n de F e inserir em E % explorados

3.2.2 Adicionar em F todos os sucessores de n que não estejam em E em ordem crescente de custo e insere n nos caminhos até o estado inicial

3.2.3 Voltar ao passo 2

Exemplo



F = [[s](0)]

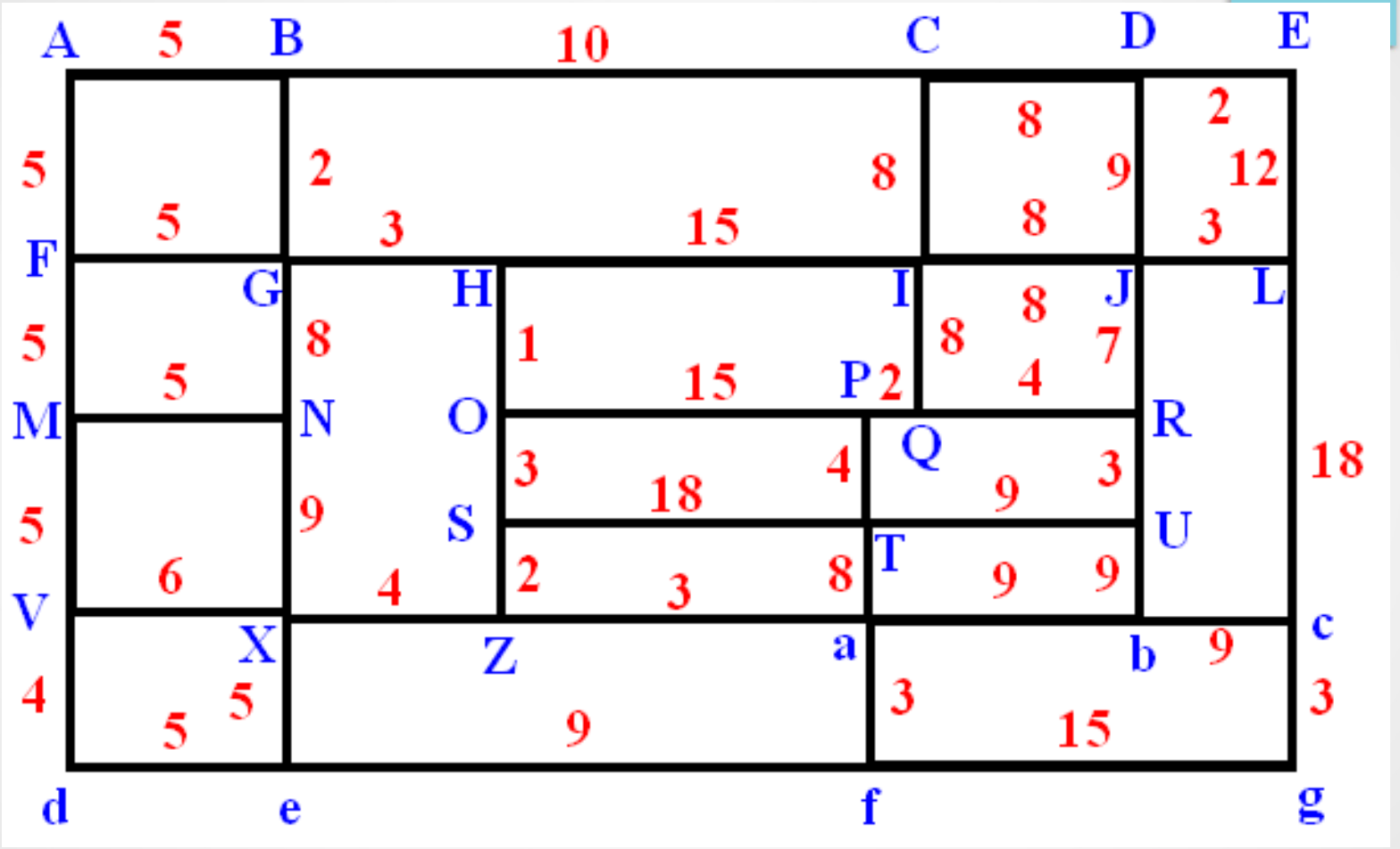
F = [[a,s](1),[b,s](5),[c,s](15)]

F = [[b,s](5),[g,a,s](11),[c,s](15)]

F =[[g,b,s](10),[g,a,s](11),[c,s](15)]

O algoritmo retorna a meta com menor custo, ou seja, g-b-s!!!

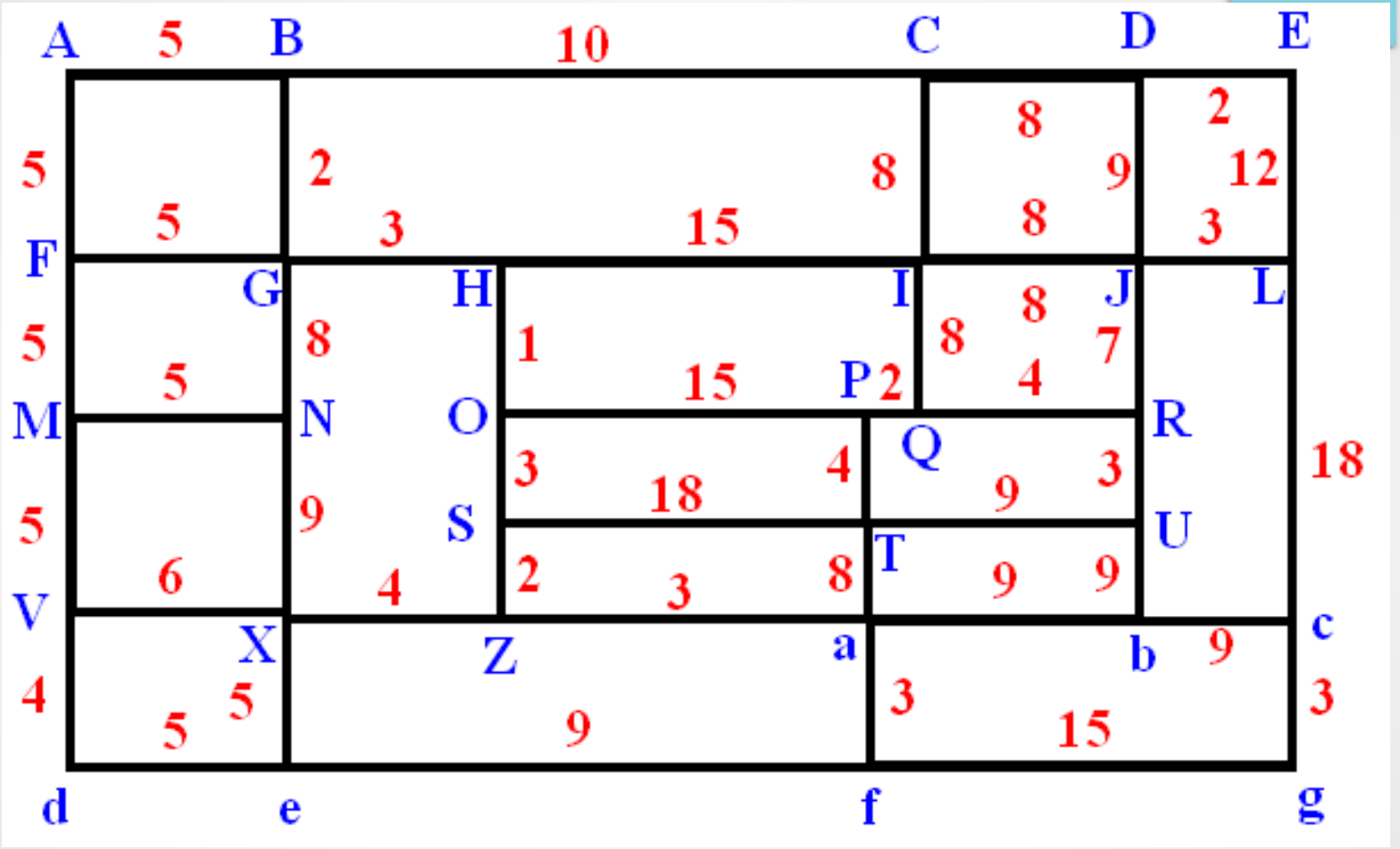
Exemplo: caminho de A - O



F={[A](0)}

E={}

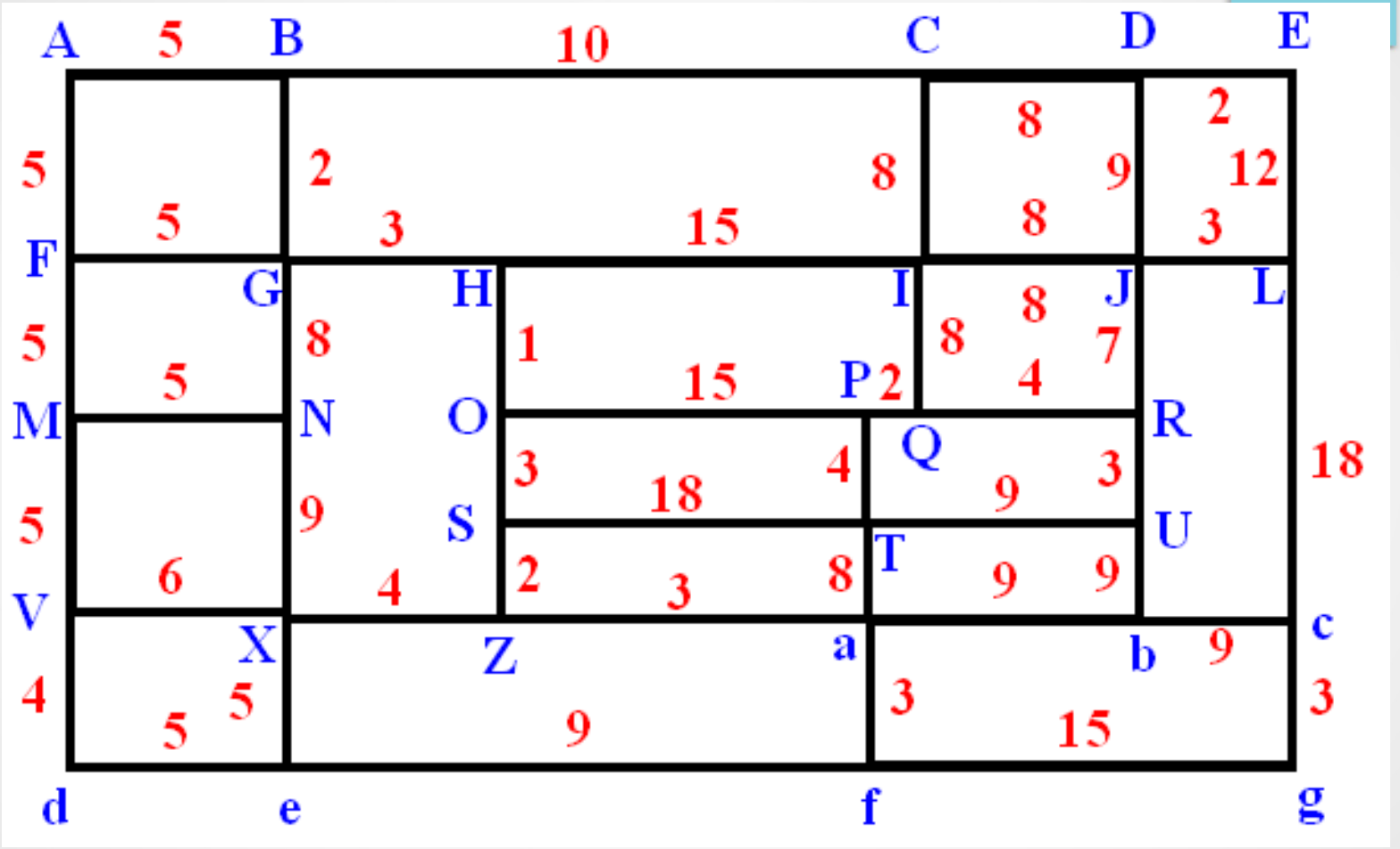
Exemplo: caminho de A - O



F={[B,A](5),[F,A](5)}

E={A}

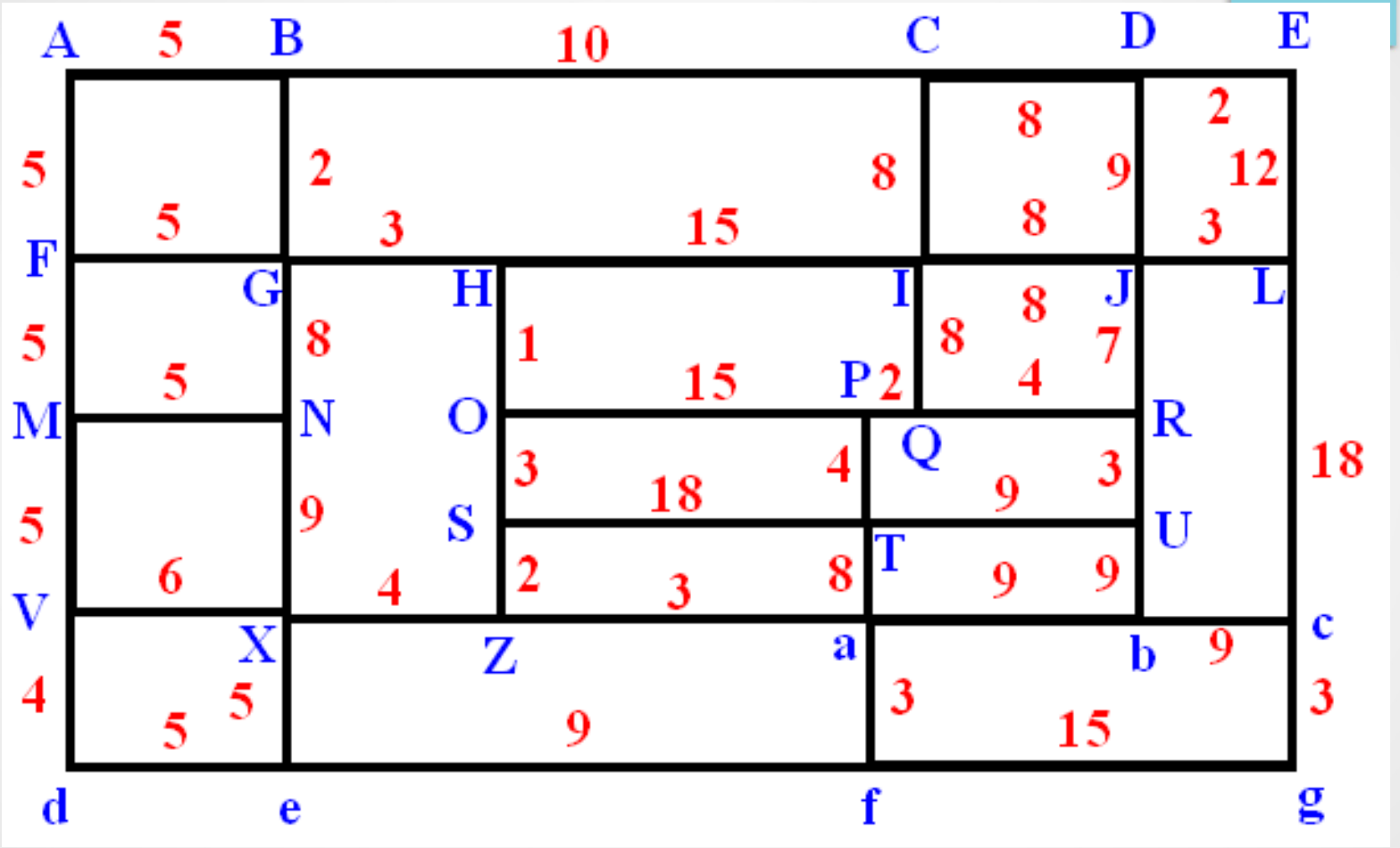
Exemplo: caminho de A - O



F={[F,A](5),[G,B,A](7),[C,B,A](15)}

E={A,B}

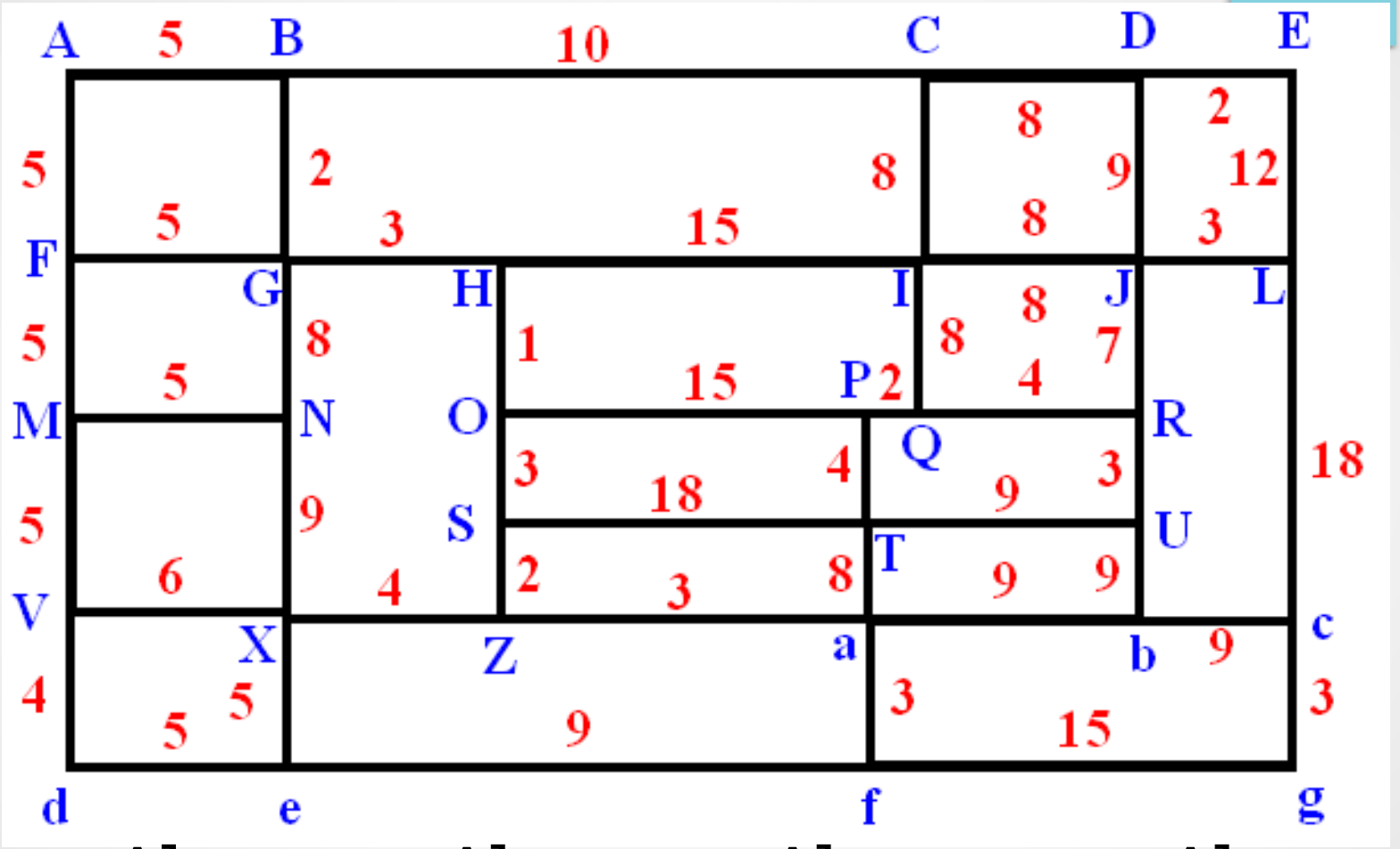
Exemplo: caminho de A - O



F={[G,B,A](7),[M,F,A](10),[G,F,A](10),[C,B,A](15)}

E={A,B,F}

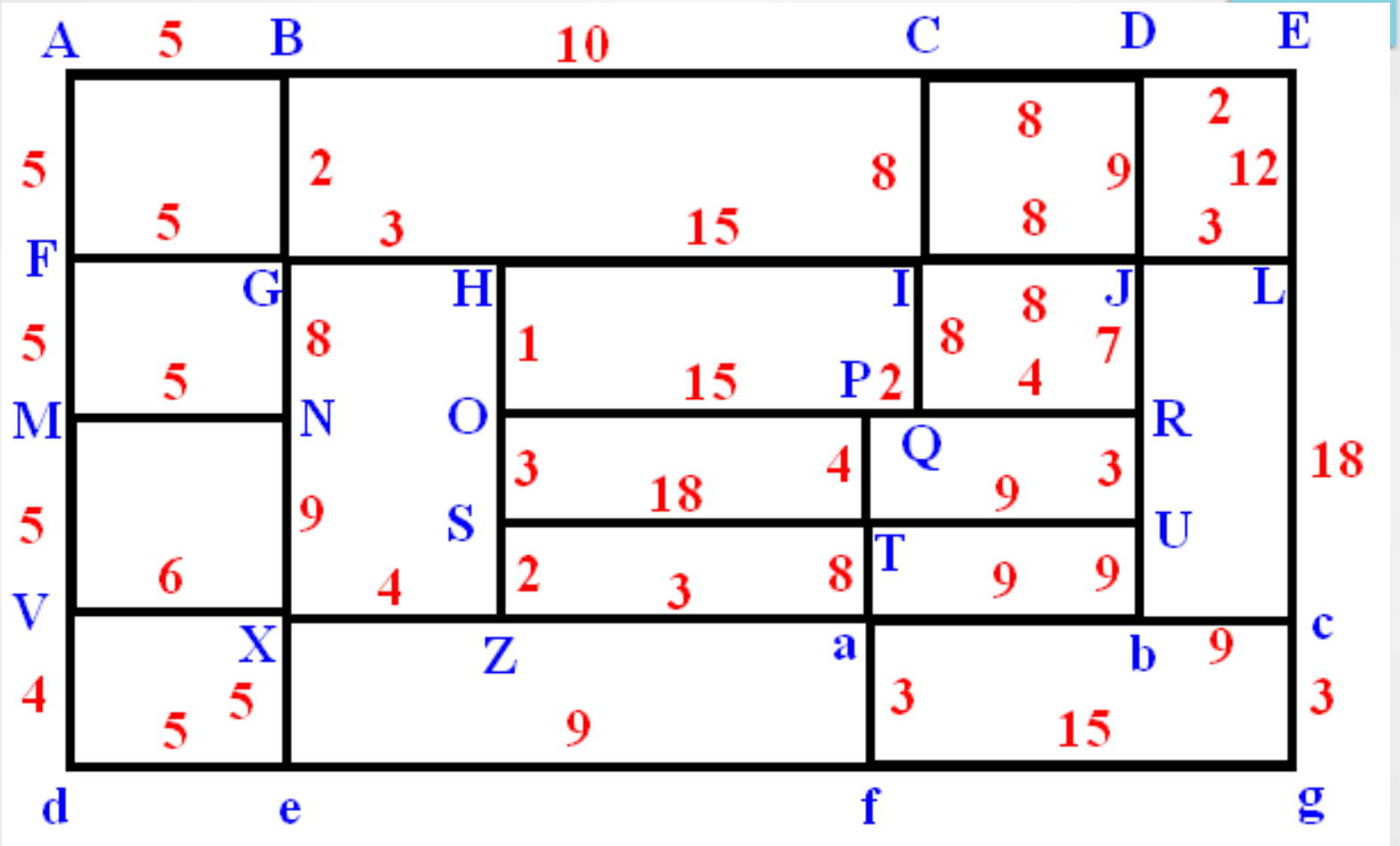
Exemplo: caminho de A - O



F={[H,G,B,A](10),[M,F,A](10),[G,F,A](10),[N,G,B,A](15),[C,B,A](15)}

E={A,B,F,G}

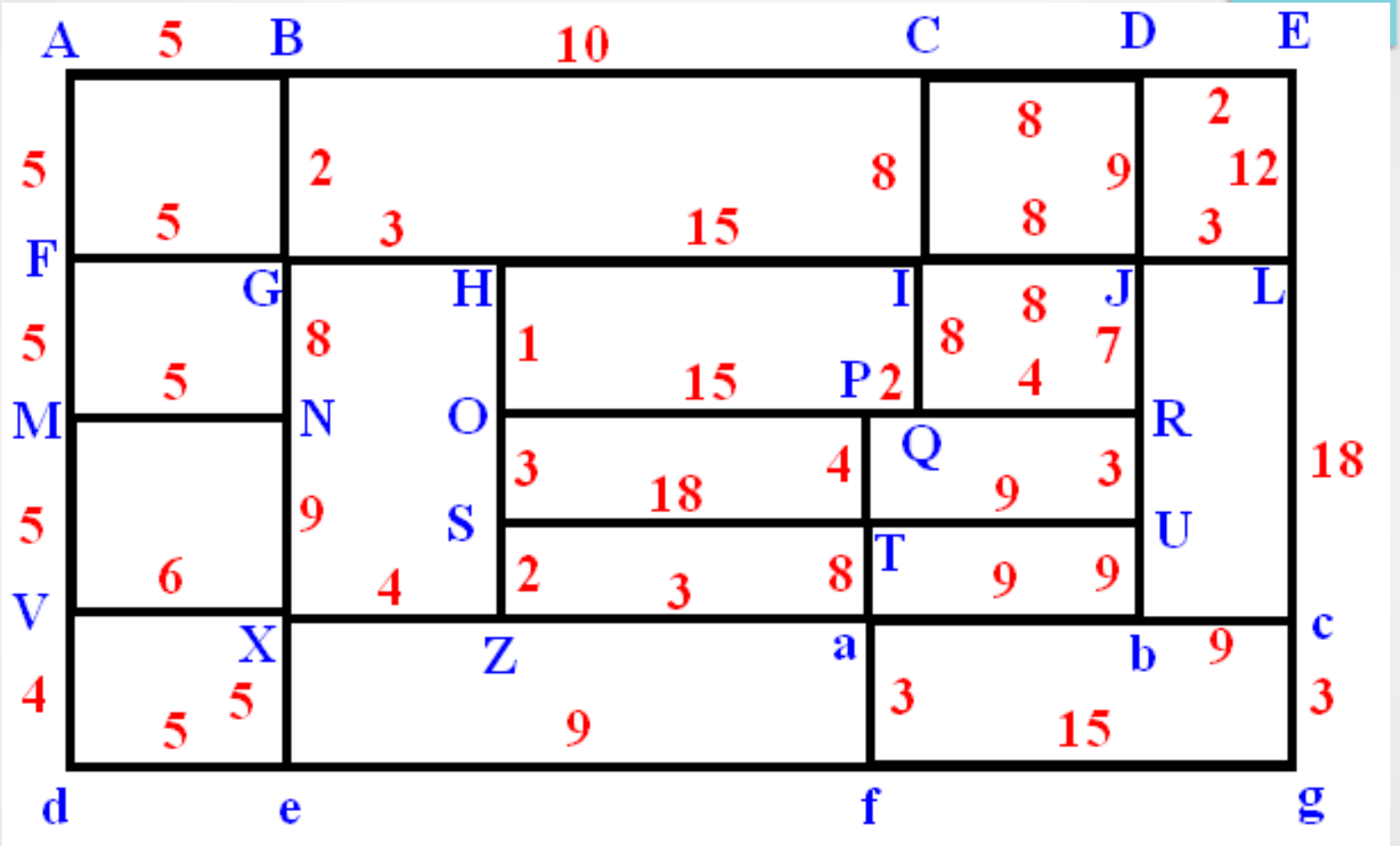
Exemplo: caminho de A - O



F={[M,F,A](10),[G,F,A](10),[O,H,G,B,A](11),[N,G,B,A](15),
[C,B,A](15),[I,H,G,B,A](25)}

E={A,B,F,G,H}

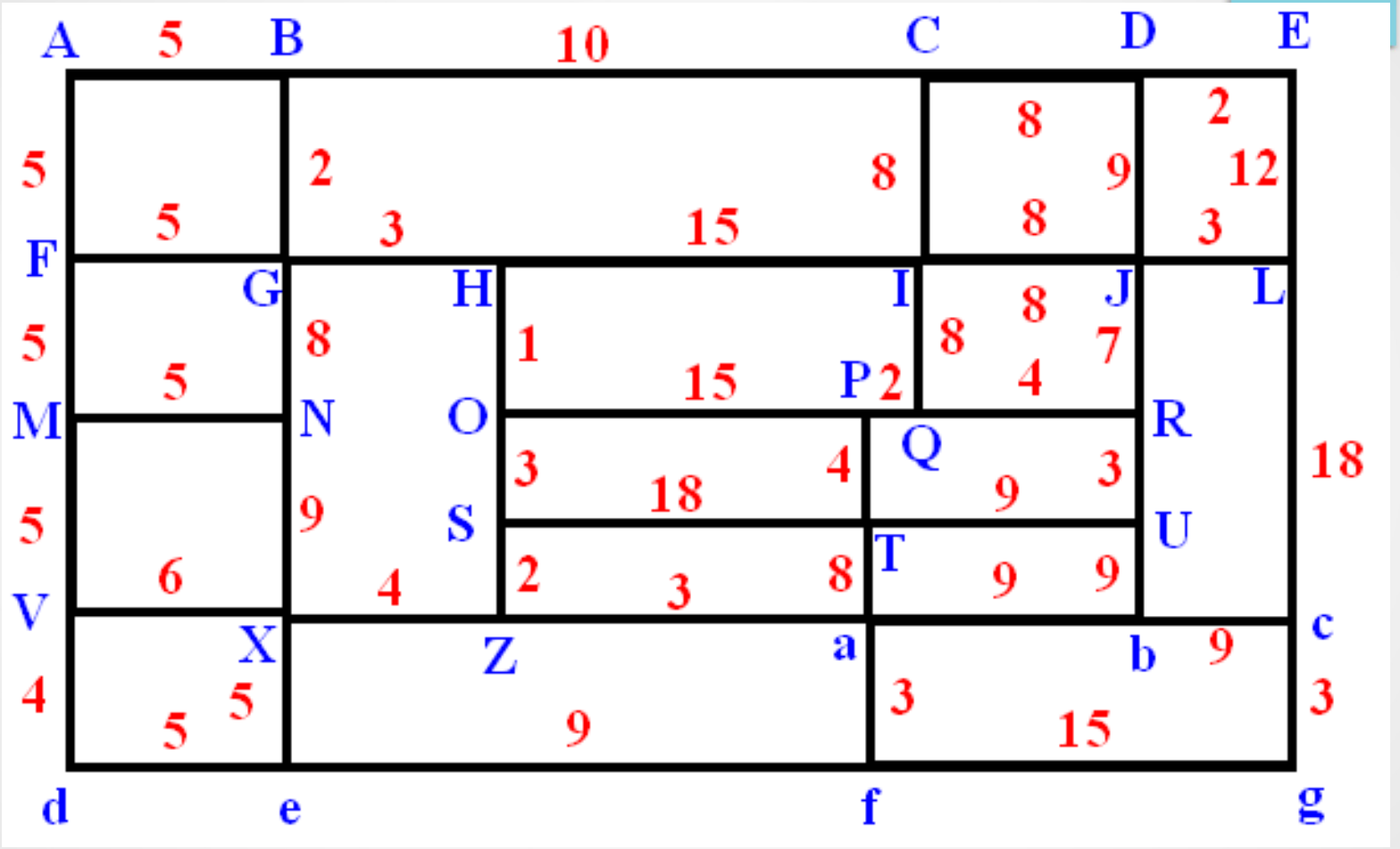
Exemplo: caminho de A - O



F={[G,F,A](10),[O,H,G,B,A](11),[N,M,F,A](15),[V,M,F,A](15),
[N,G,B,A](15),[C,B,A](15),[I,H,G,B,A](25)}

E={A,B,F,G,H,M}

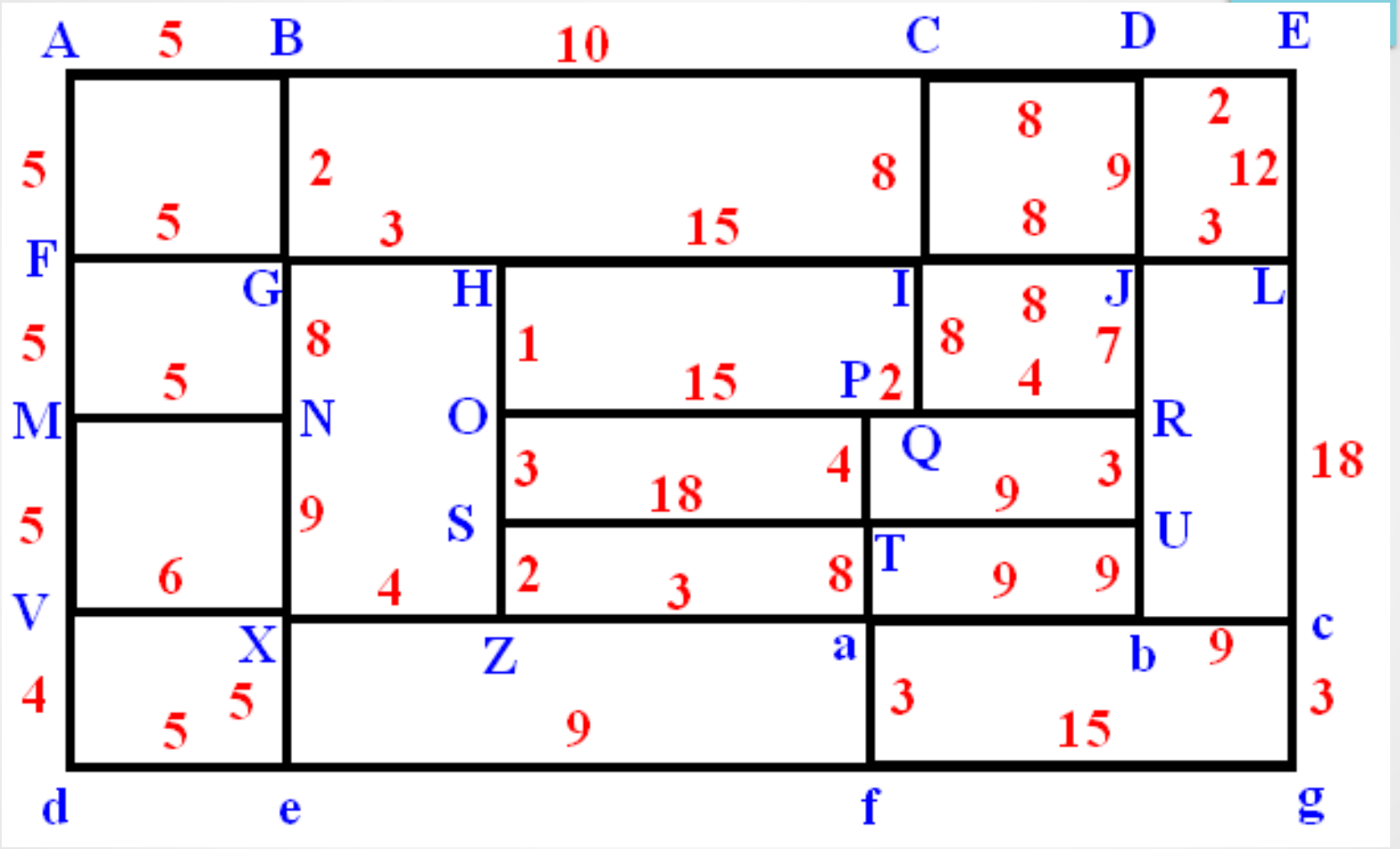
Exemplo: caminho de A - O



F={[O,H,G,B,A](11),[H,G,F,A](13),[N,M,F,A](15),[V,M,F,A](15),
[N,G,B,A](15),[C,B,A](15),[N,G,F,A](18),[I,H,G,B,A](25)}

E={A,B,F,G,H,M}

Exemplo: caminho de A - O



- A busca encontra o caminho [O,H,G,B,A](11) de A até O

Características da BCU

- Completo?? Sim.
- Tempo?? $O(b^d)$
- Espaço?? $O(b^d)$
- Admissível?? Sim.

Alguns tipos de busca cega

- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Profundidade Iterativa

Busca em Profundidade

- Consiste em procurar pelo(s) objetivo(s) pelo estado mais recentemente encontrado pelo mecanismo de busca
- Novos estados são inseridos no início da lista de estados de fronteira (F)
- Para evitar repetição de caminhos, usa lista de espaços explorados (E)
- Tem esse nome porque quando aplicado em árvores (grafos conexos e acíclicos), a busca é feita de cima para baixo!!

Algoritmo BL

1 Inserir os nós iniciais na lista de busca F % fronteira

2 Se F é vazio

2.1 Então a busca não foi bem sucedida

3 Senão seja n o primeiro estado de F

3.1 Se n é um estado meta então

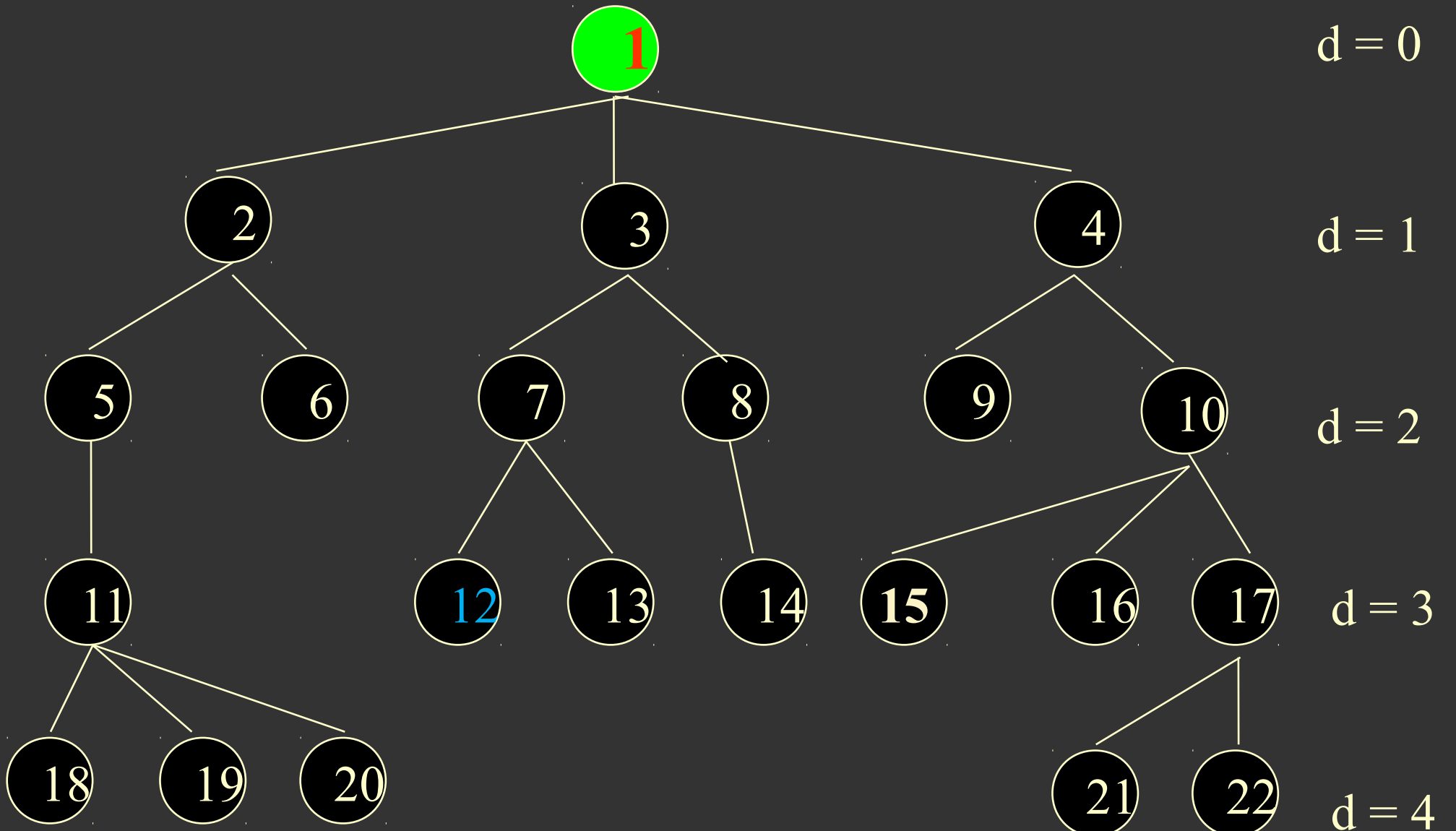
3.1.1 Retornar n

3.2 Senão

3.2.1 Remover n de F e inserir em E % explorados

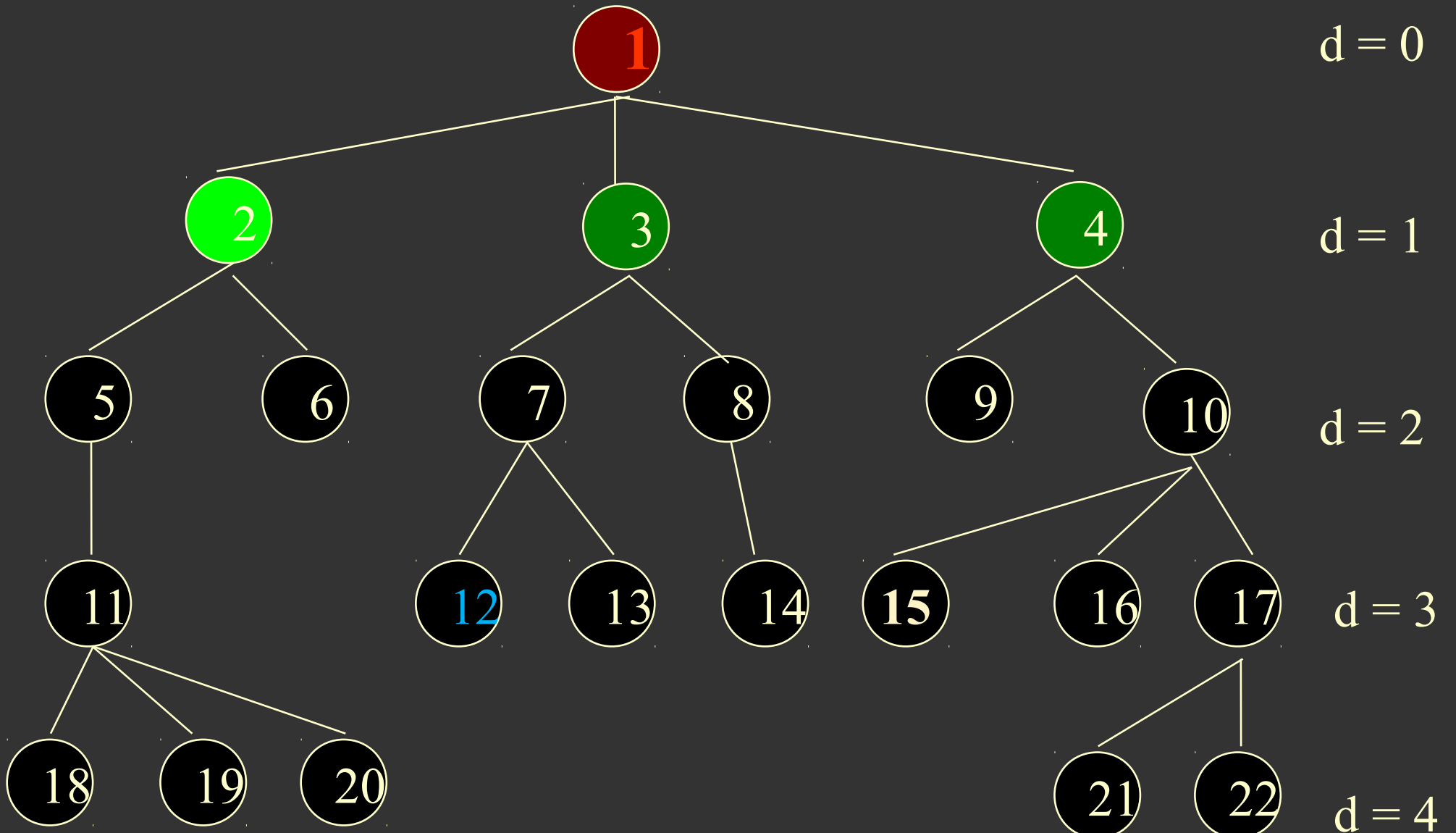
3.2.2 Adicionar no início de F todos os sucessores de n que não estejam em F ou E junto com n e o caminho até o estado inicial

3.2.3 Voltar ao passo 2



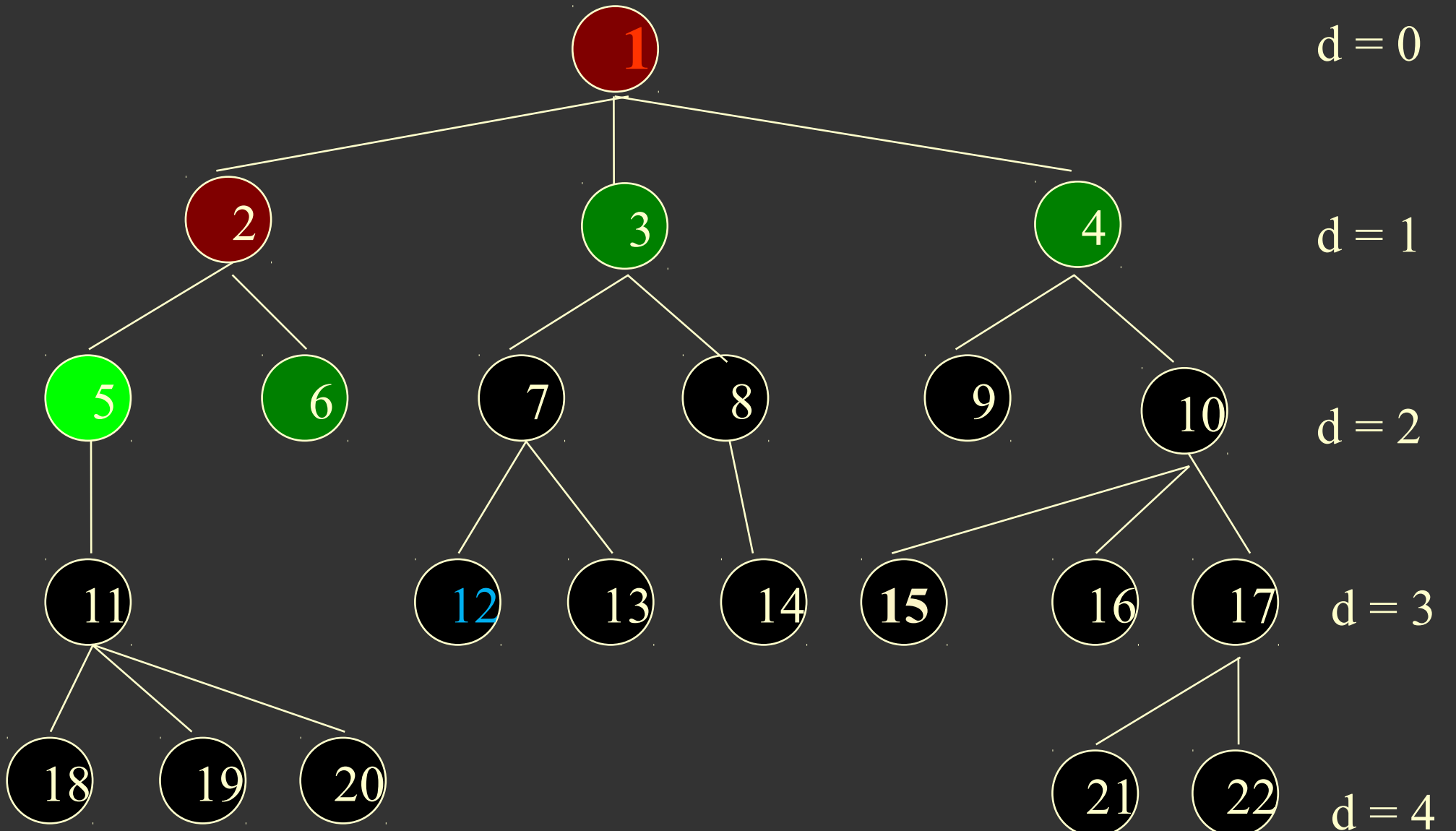
F={[1]}

E={}



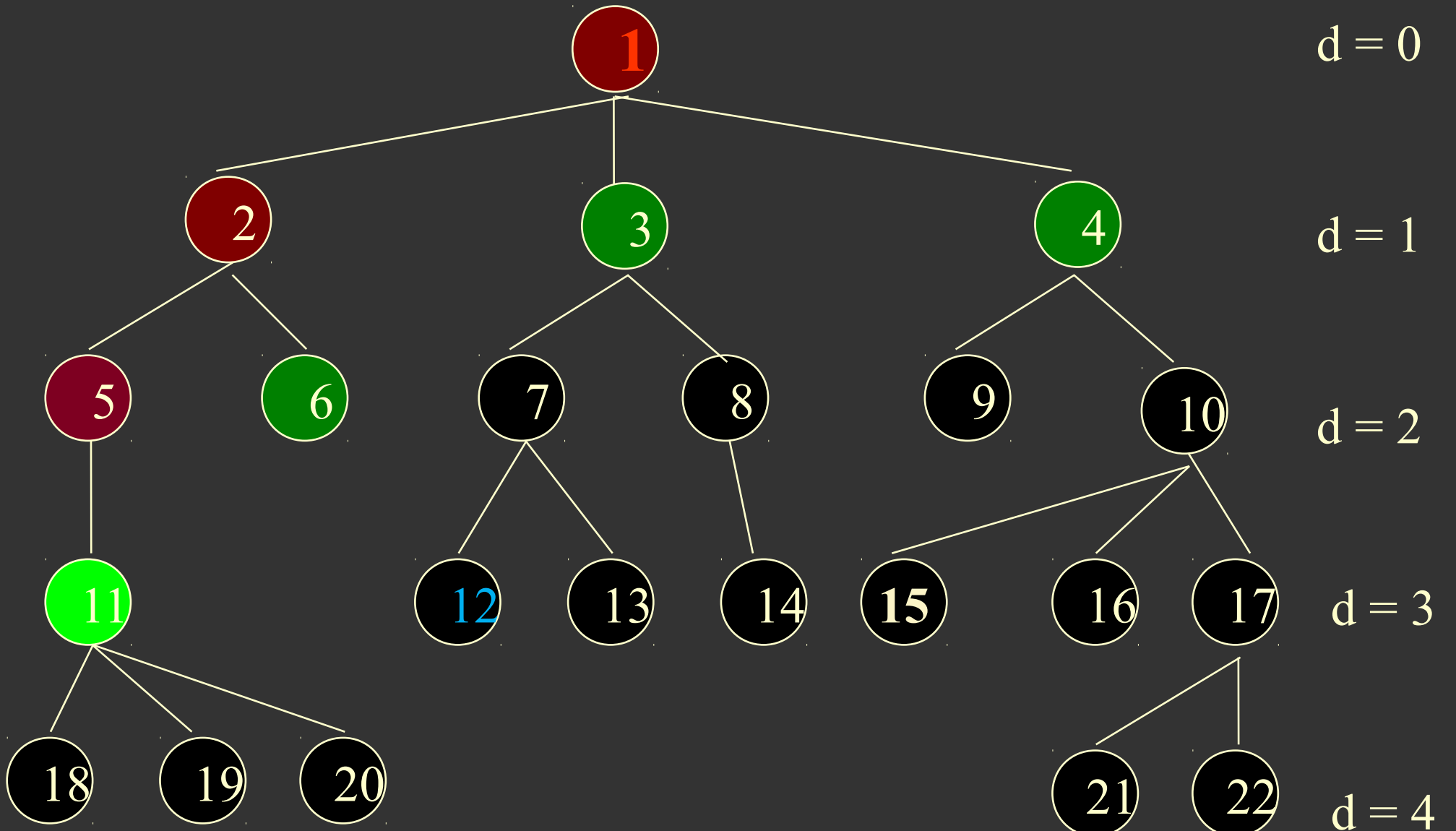
F={[2,1],[3,1],[4,1]}

E={1}



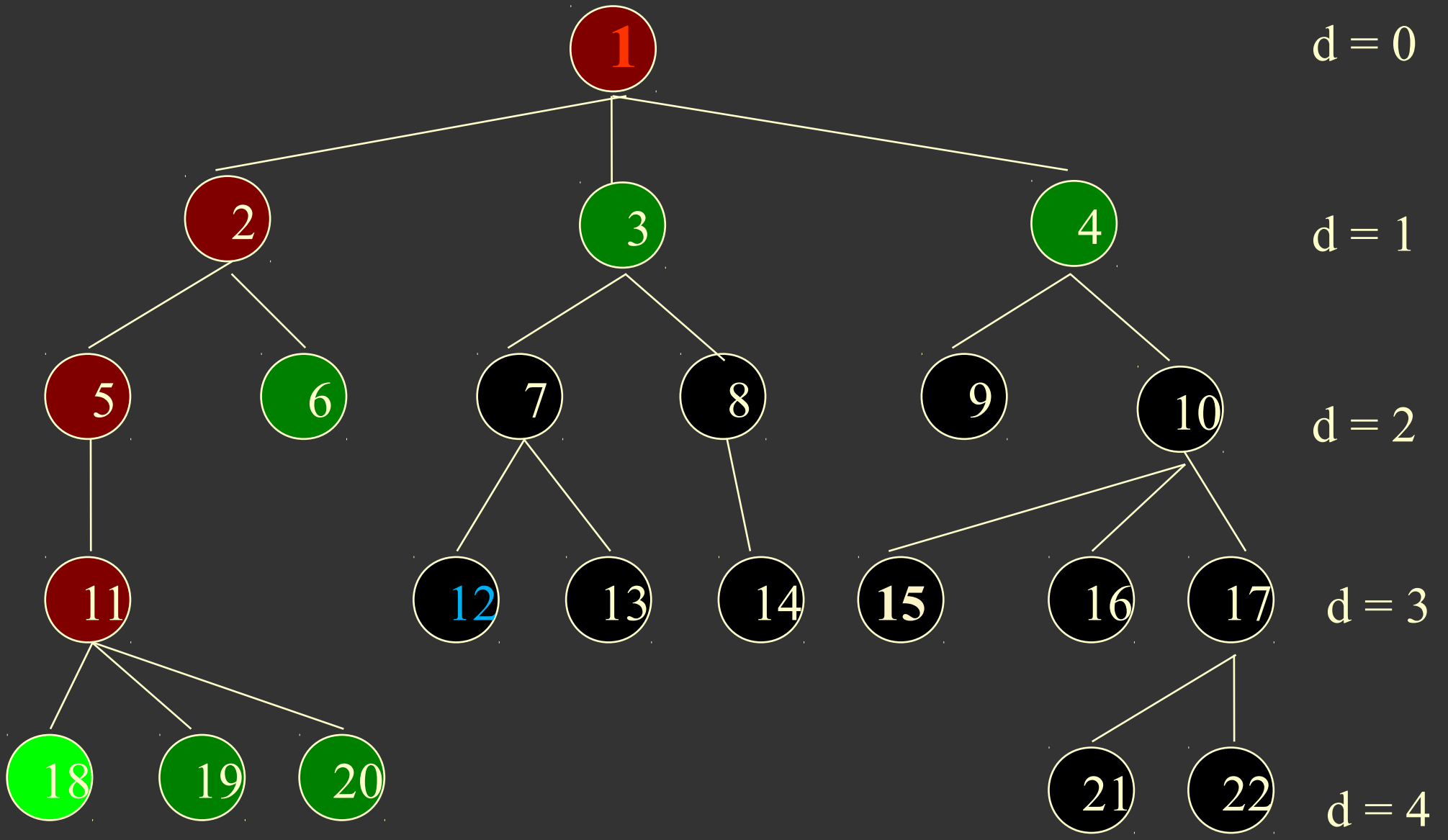
F={[5,2,1],[6,2,1],[3,1],[4,1]}

E={1,2}



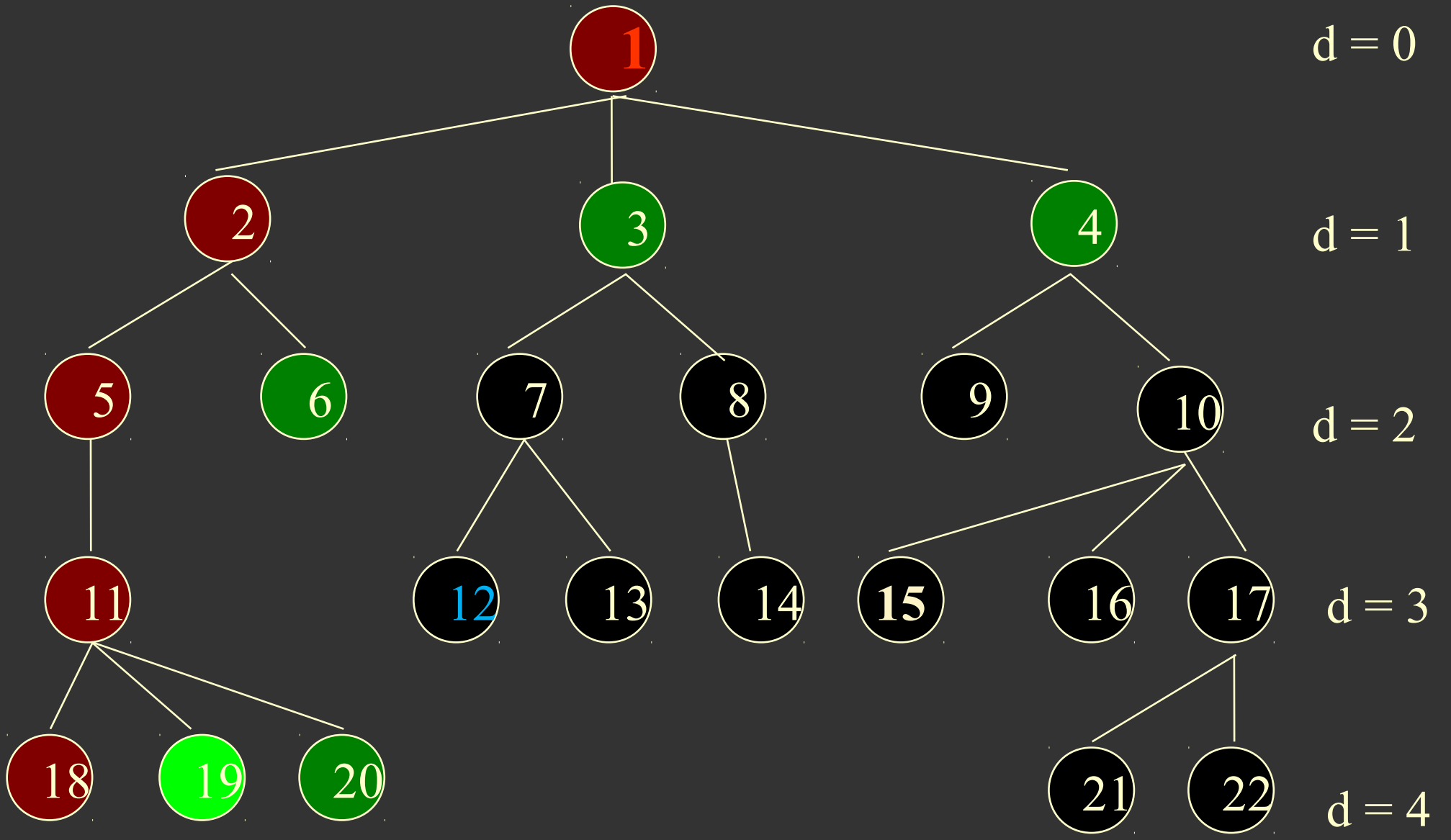
F={[11,5,2,1],[6,2,1],[3,1],[4,1]}

E={1,2,5}



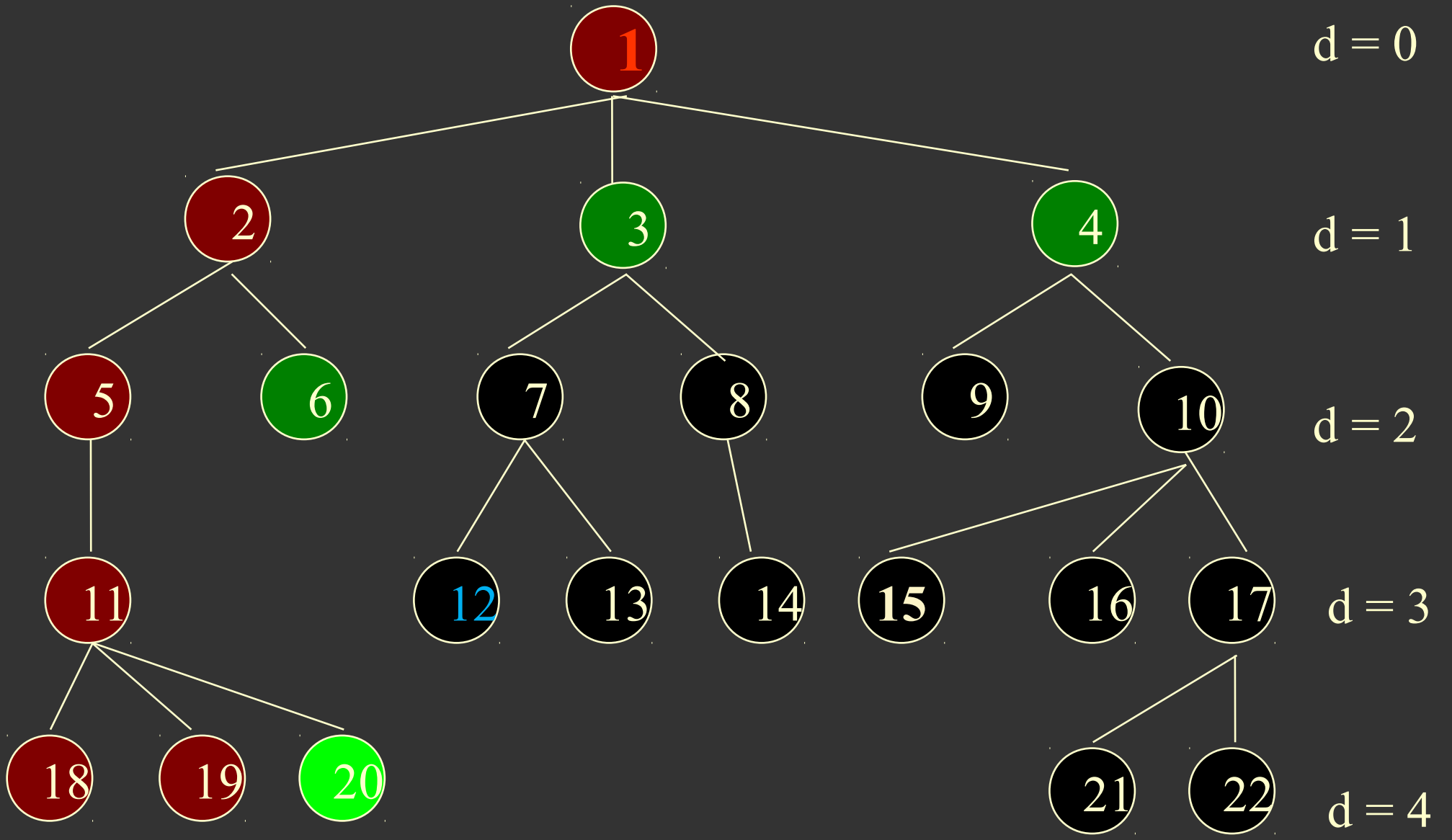
F={[18,11,5,2,1],[19,11,5,2,1],[20,11,5,2,1],
[6,2,1],[3,1],[4,1]}

E={1,2,5,11}



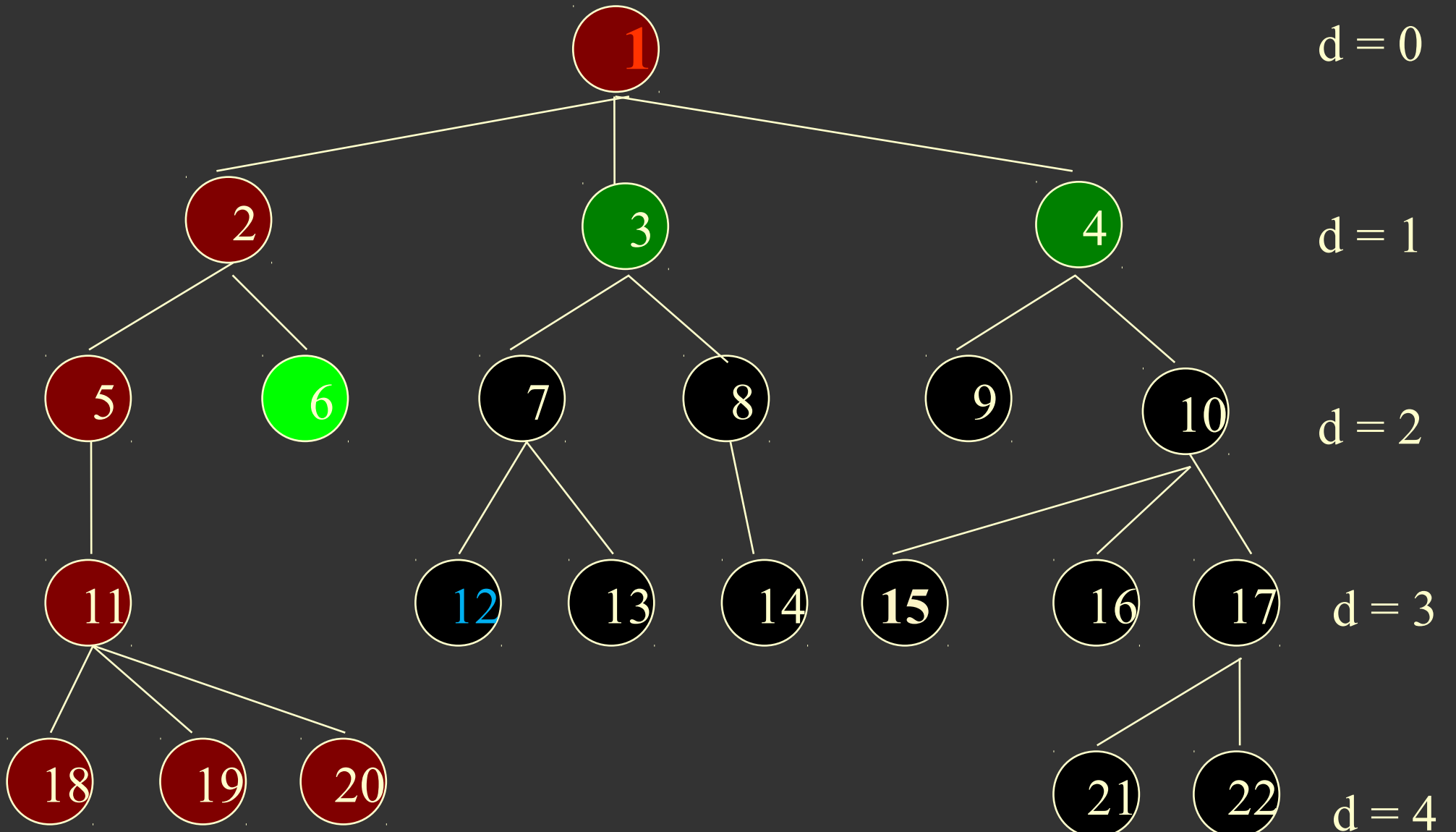
F={[19,11,5,2,1],[20,11,5,2,1],[6,2,1],[3,1],[4,1]}

E={1,2,5,11,18}



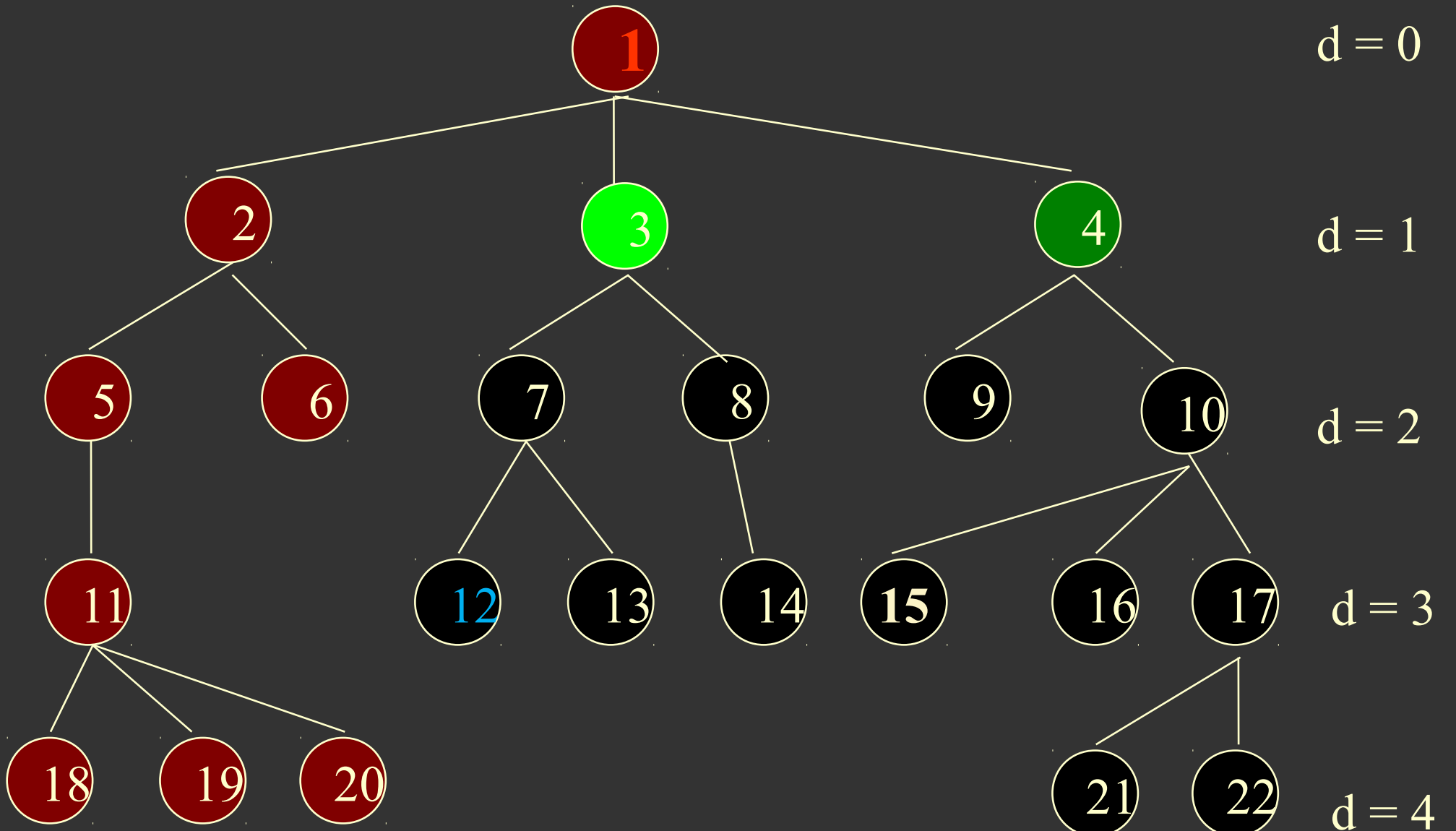
F={[20,11,5,2,1],[6,2,1],[3,1],[4,1]}

E={1,2,5,11,18,19}



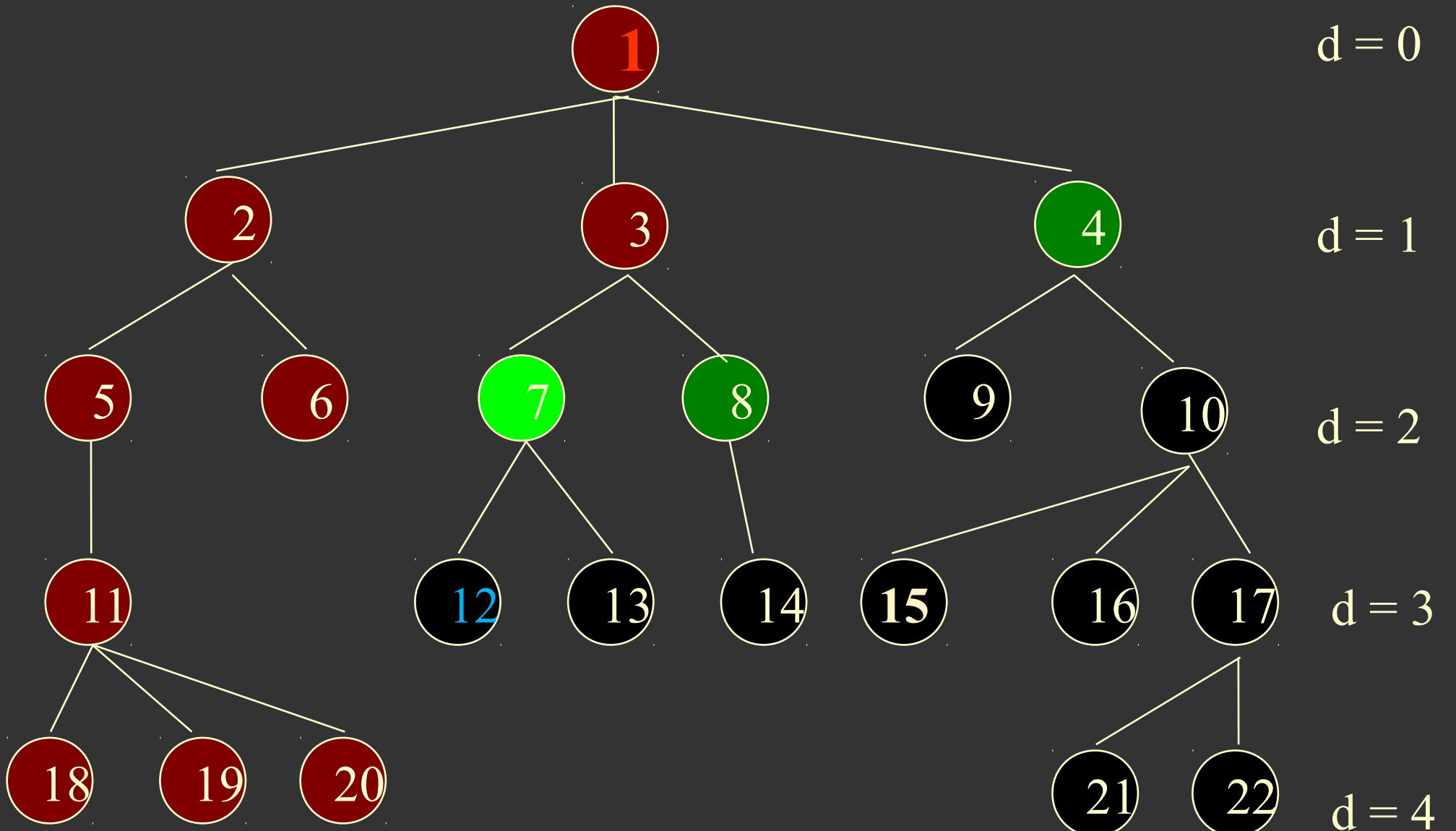
F={[6,2,1],[3,1],[4,1]}

E={1,2,5,11,18,19,20}



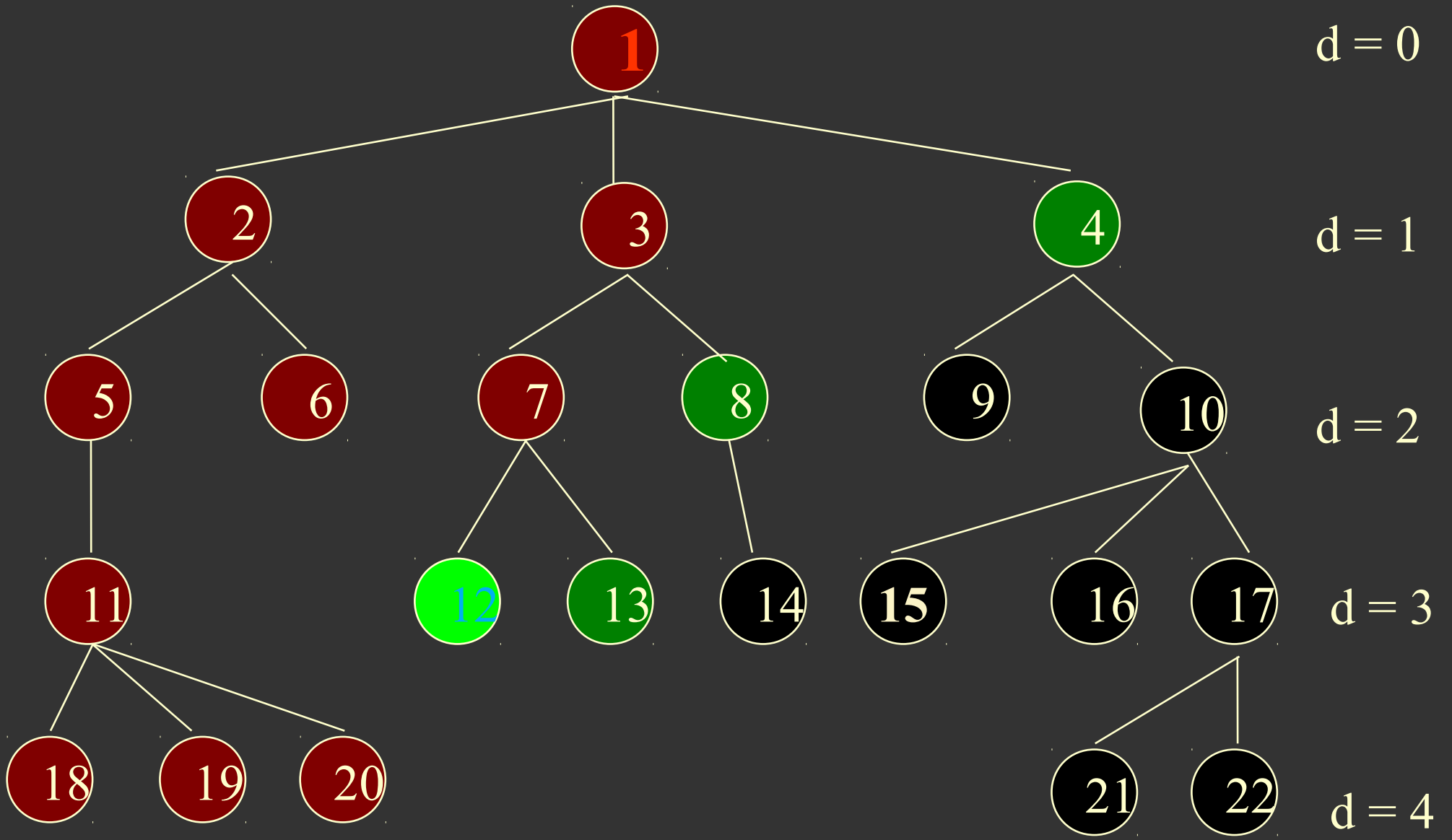
F={[3,1],[4,1]}

E={1,2,5,11,18,19,20,6}



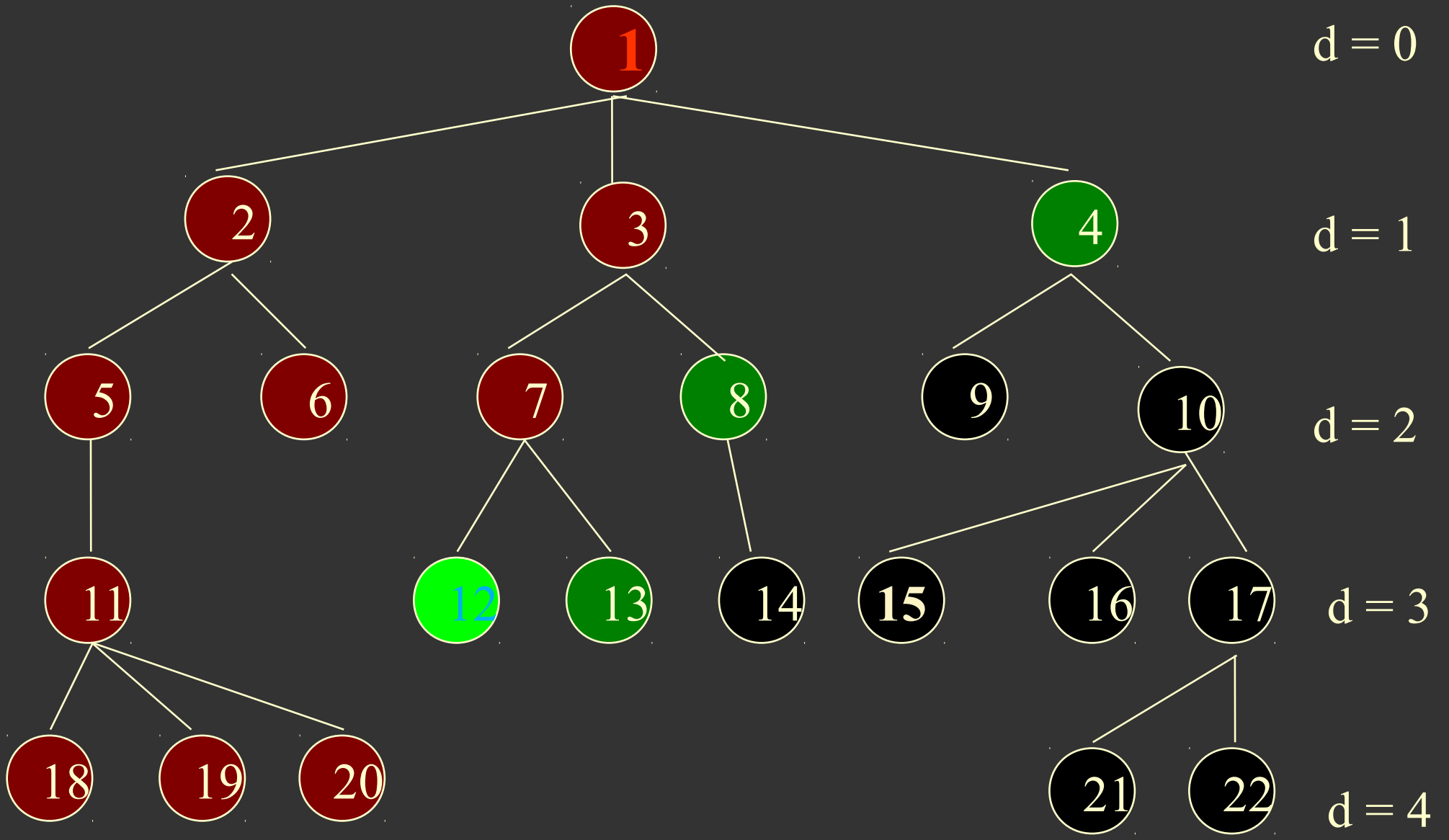
F={[7,3,1],[8,3,1],[4,1]}

E={1,2,5,11,18,19,20,6,3}



F={[12,7,3,1],[13,7,3,1],[8,3,1],[4,1]}

E={1,2,5,11,18,19,20,6,3,7}



- Objetivo encontrado! = [12,7,3,1]

Propriedades da BP

- Completo?? Sim se o espaço de estados for de profundidade finita. Não caso contrário.
- Tempo?? $O(b^m)$. Bom se m é igual à *d* ou ruim se m é maior que *d*.
- Espaço?? $O(bm)$ linear em *b* e *m*!!!!
- Admissível?? Não, pois não encontra sempre o menor caminho.

Código BP em Prolog

- Começamos encapsulamento da busca

```
%solucao por busca em profundidade (bp)

solucao_bp(Inicial,Solucao) :- bp([],Inicial,Solucao).
```

- a função bp recebe o caminho para o estado, o estado a ser explorado e retorna a solução.
- Se estado for a meta, monta e retorna a solução com o estado meta e seu caminho

```
%Se o primeiro estado da lista é meta, retorna a meta

bp(Caminho,Estado,[Estado|Caminho]) :- meta(Estado).
```

Código BP em Prolog

- Se a busca em profundidade não encontra a meta, então inserimos o estado no caminho (cabeça do caminho) e procuramos nos estados sucessores (descendentes)

```
%se falha, coloca o no caminho e continua a busca

bp(Caminho,Estado,Solucao) :- s(Estado,Sucessor),
not(pertence(Sucessor,[Estado|Caminho])),

    bp([Estado|Caminho],Sucessor,Solucao).
```

- Mais simples que a BL por inserir estados no começo da lista e uso do **retrocesso**!

Alguns tipos de busca cega

- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Profundidade Iterativa

Busca em profundidade limitada

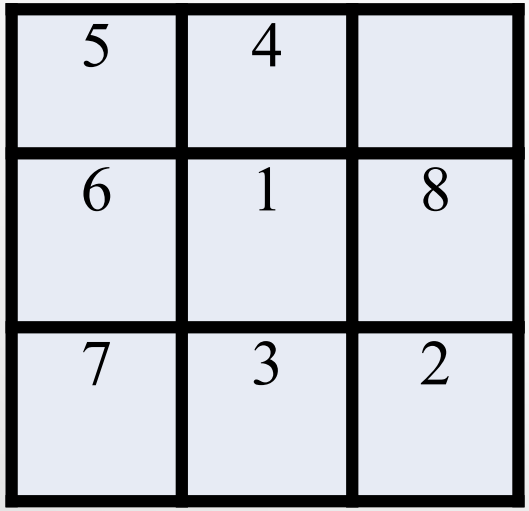
- Na tentativa de tentar solucionar problemas profundidade alta, se aplica um limite *l* na BP
- Assume-se que os nós à profundidade *l* não possuem sucessores. (Qual o problema?)
- Completo?? Sim se *l* >= *d*
- Tempo?? O(b^l)
- Espaço?? O(*bl*)
- Admissível?? Não.

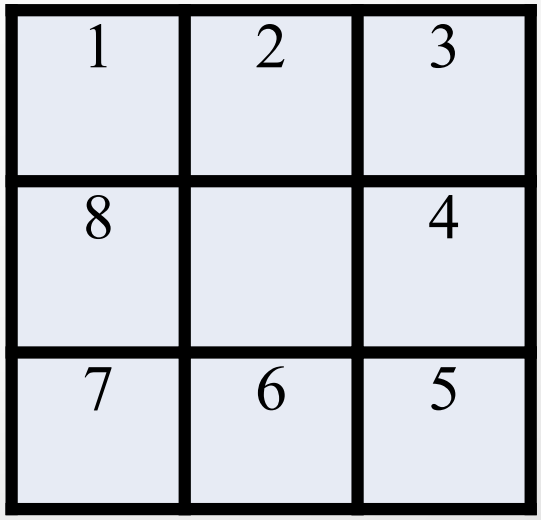
Busca em profundidade iterativa

- Chama BP limitada iterativamente.
- Escapa da necessidade de escolher o melhor limite de profundidade, pois tenta todos os limites possíveis após o limite definido t: primeiro t, depois t+1, então t+2,...
- Combina as vantagens da BP com a BL
- Tempo?? O(b^d) no pior caso
- Espaço?? O(bd)
- Completo?? Sim.
- Admissível?? A partir de t, sim.

Aplicações

- São inúmeras as aplicações para busca em espaço de estados
- 8-Puzzle
  - Quebra cabeças com 8 números
  - Só pode trocar o espaço vazio com um número adjacente à ele





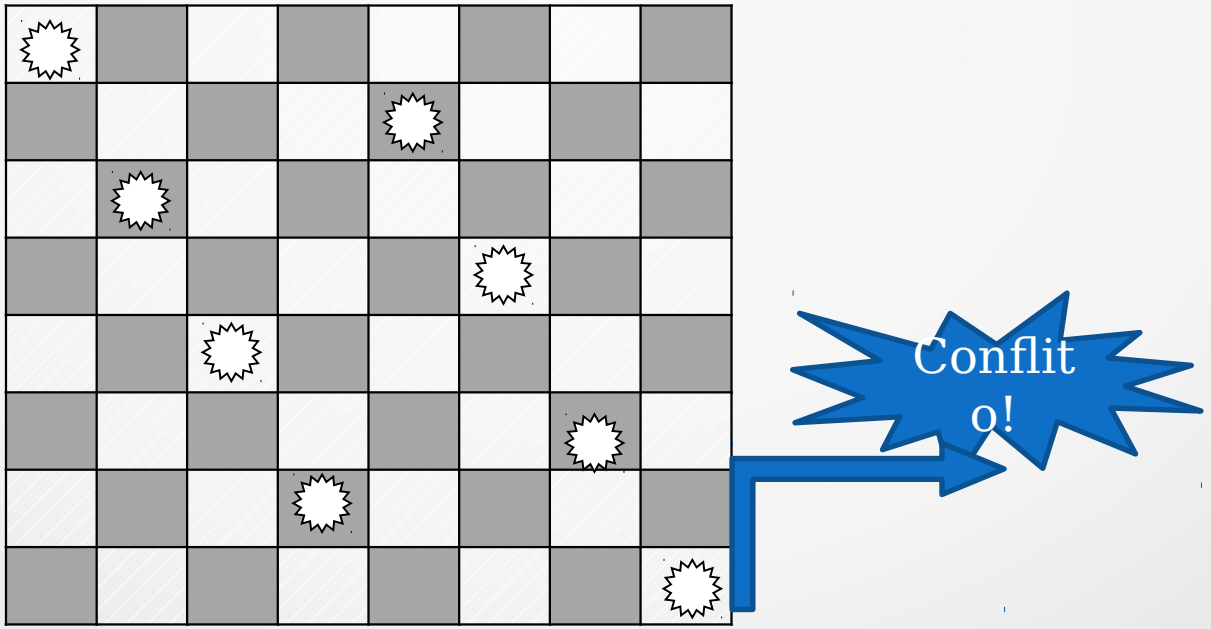
8-Puzzle

- Modelagem

Estados:	especifica localização de cada uma das oito peças - é o próprio tabuleiro
Operadores:	espaço move à direita espaço move à esquerda espaço move para cima espaço move para baixo
Objetivo:	estado final definido
Custo do caminho:	cada passo tem custo 1

Oito rainhas

- Colocar 8 rainhas em um tabuleiro de xadrez de forma que uma não possa atacar a outra. (Uma rainha ataca qualquer peça na mesma linha, coluna ou diagonal)



Oito rainhas

Formulação

Estados:	arranjo de rainhas no tabuleiro, nenhuma sendo atacada
Operadores:	adicionar uma rainha, em qualquer posição de modo que não seja atacada pelas outras
Objetivo:	8 rainhas no tabuleiro, nenhuma atacada
Custo do caminho:	0 (só interessa a situação final)

Outra Formulação

Operadores:	adicionar uma rainha, na coluna vazia mais à esquerda, de modo que não seja atacada pelas outras

Jarros

- Você tem duas jarras, uma de 4 litros e outra de 3 litros. Nenhuma delas tem qualquer marcação de medidas. Há uma bomba que pode ser usada para encher as jarras com água. Como é que você consegue obter exatamente 2 litros de água?

Fazendeiro-Lobo-Ovelha-Repolho

- Um fazendeiro, uma ovelha, um repolho e um lobo encontram-se do lado norte de um rio. O fazendeiro possui uma barca que ele pode utilizar para atravessar a ovelha ou o lobo ou o repolho para lado sul do rio. A ovelha não pode ficar na mesma margem que o repolho e o lobo não pode ficar na mesma margem que a ovelha, a não ser que o fazendeiro esteja presente. Modele o problema e faça com que o Prolog consiga transferir o fazendeiro, o ovelha e o repolho para a margem sul.

Exercícios

- Resolva os problemas exemplos dados em aula utilizando busca cega
  - Fazer a modelagem dos problemas
  - Código das buscas e manipulação de listas foi dado em aula
    - É copiar e colar!

Prolog - Jarros

```
%Regras de sucessão de estados
%enche o jarro de 4 litros
s([0,Y],[4,Y]).
%enche o jarro de 3 litros
s([X,0],[X,3]).
%passa agua do jarro de 4 litros para o de 3 litros ate ele encher
s([X,Y],[X2,3]):- Y < 3, X > (3-Y), X2 is X-(3-Y).
%passa agua do jarro de 4 litros para o de 3 litros ate ficar vazio
s([X,Y],[0,Y2]):- Y < 3, X < (3-Y), X > 0, Y2 is X+Y.
%passa agua do jarro de 3 litros para o de 4 litros ate ele encher
s([X,Y],[4,Y2]):- X < 4, Y > (4-X), Y2 is Y-(4-X).
%passa agua do jarro de 3 litros para o de 4 litros ate ficar vazio
s([X,Y],[X2,0]):- X < 4, Y < (4-X), Y > 0, X2 is X+Y.
%esvazia o jarro de 4 litros
s([_,Y],[0,Y]).
%esvazia o jarro de 3 litros
s([X,_],[X,0]).

%Define o estado meta
meta(Estado) :- pertence(2,Estado).
```

Prolog – Fazendeiro (parte1)

```
% Assume que o estado seja representado por uma lista cujas posições
% são fazendeiro, lobo, ovelha e repolho
% neste problema, o estado inicial é fixo, ou seja,
% [norte,norte,norte,norte]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Regras de sucessão de estados
%fazendeiro atravessa o rio com lobo
s([Saida,Saida,O,R],[Chegada,Chegada,O,R]) :-
permitido_ir_de([Saida,Saida,O,R],[Chegada,Chegada,O,R]).
%fazendeiro atravessa o rio com a ovelha
s([Saida,L,Saida,R],[Chegada,L,Chegada,R]) :-
permitido_ir_de([Saida,L,Saida,R],[Chegada,L,Chegada,R]).
%fazendeiro atravessa o rio com o repolho
s([Saida,L,O,Saida],[Chegada,L,O,Chegada]) :-
permitido_ir_de([Saida,L,O,Saida],[Chegada,L,O,Chegada]).
%fazendeiro atravessa o rio sozinho
s([Saida,L,O,R],[Chegada,L,O,R]) :- permitido_ir_de([Saida,L,O,R],
[Chegada,L,O,R]).
```

Prolog – Fazendeiro (parte2)

```
%regra que define se é permitido atravessar o rio
permitido_ir_de([F1,_,_,_],[F2,L2,O2,R2]) :- opostas(F1,F2),
nao_oferece_perigo(F2,L2,O2,R2).

%regra que define estados que oferecem perigo e não são permitidos
%não há perigo se o fazendeiro estiver com a ovelha ou
nao_oferece_perigo(FO,_,FO,_).
%não há perigo se a ovelha estiver sozinha em uma das margens
nao_oferece_perigo(FLR,FLR,O,FLR) :- opostas(FLR,O).

%preciso definir quais são as margens opostas
opostas(sul,norte).
opostas(norte,sul).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%define estado inicial
inicio([norte,norte,norte,norte]).

%define o estado meta
meta([sul,sul,sul,sul]).
```