分枝限定法

分枝限定法

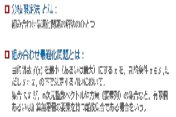
分枝限定法 とは：

組み合わせ最適化問題の解法のひとつ

組み合わせ最適化問題とは：

目的関数 $f(x)$ を最小（あるいは最大）にする x を，制約条件 $x \in S$,ただし $S \subset X$, の下で決定する問題において，

集合 S, X が，n次元整数ベクトルや方策（要素列）の集合など，有限個あるいは加算無限個の要素を持つ離散集合である場合をいう。



例：配分問題

条件

$$\sum_{i=1}^{N} x_i = K(const.), x_i > 0$$ なる整数

の下で，

$$f(x_1, x_2, \cdots, x_N) = \sum_{i=1}^{N} f_i(x_i)$$

の最大化を行う問題。

配分問題の例）

総量 K の資源と，N個の工場があり，工場 i は資源 x_i をもとに収益 $f_i(x_i)$ をあげることができるものとする。このとき，収益の合計を最大化する資源の分配の方法を求める。

例）$K = 5,\ \ N = 3$ の配分問題：

条件 $\sum_{i=1}^{3} x_i = 5, x_i > 0$ なる整数

の下で，$f = \sum_{i=1}^{3} f_i(x_i)$ の最大化を行う。

解法のひとつ： 全ての可能性を列挙する。

$x_1 = 1, 2, 3$ を選べる

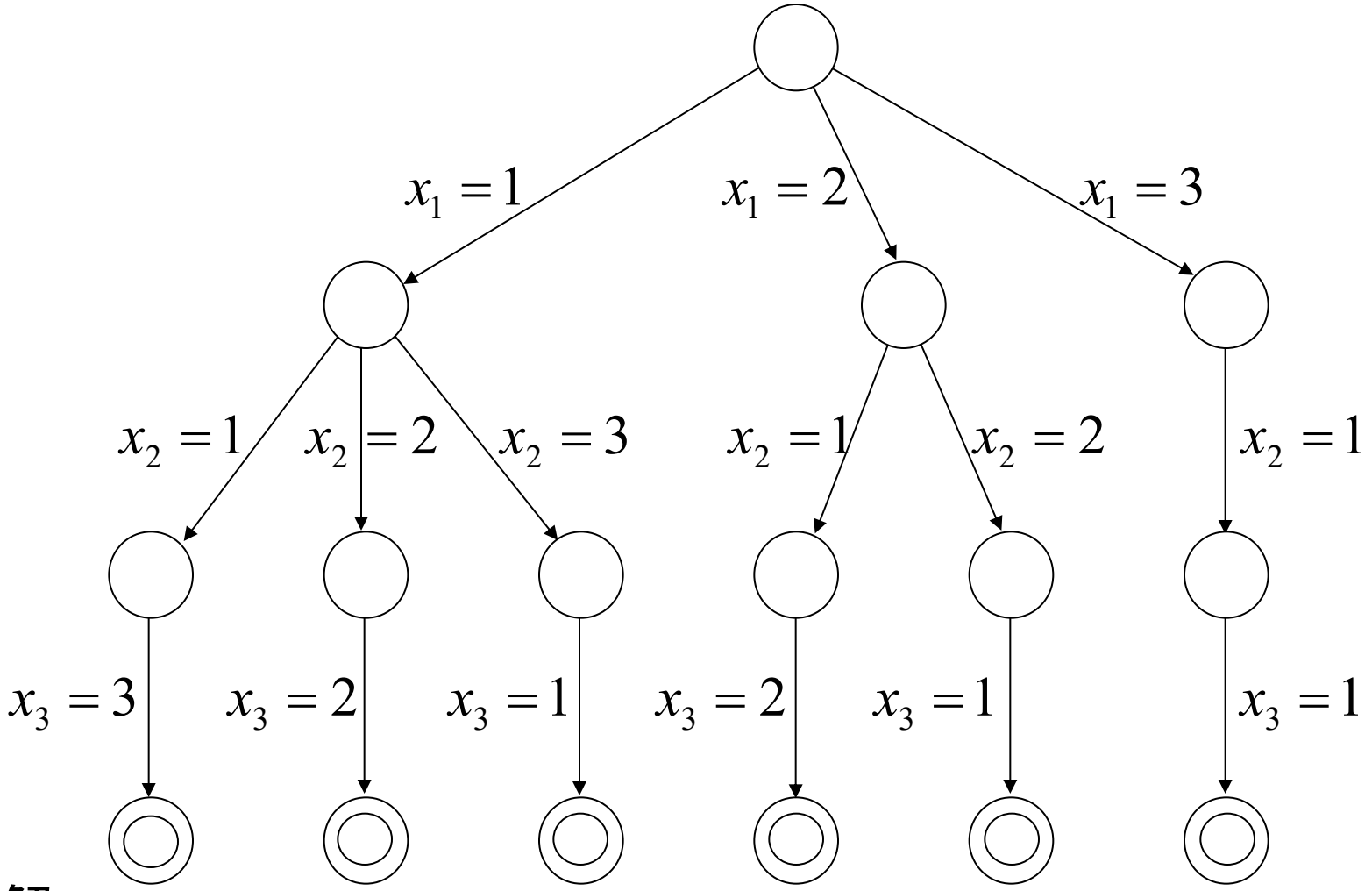
$x_1 = 1$ のとき，$x_2 = 1, 2, 3$ を選べる

$x_1 = 1,\ x_2 = 1$のとき，$x_3 = 3$ を選べる

$x_1 = 1,\ x_2 = 2$のとき，$x_3 = 2$ を選べる

：

列挙図



許容解

列挙図における枝の解釈

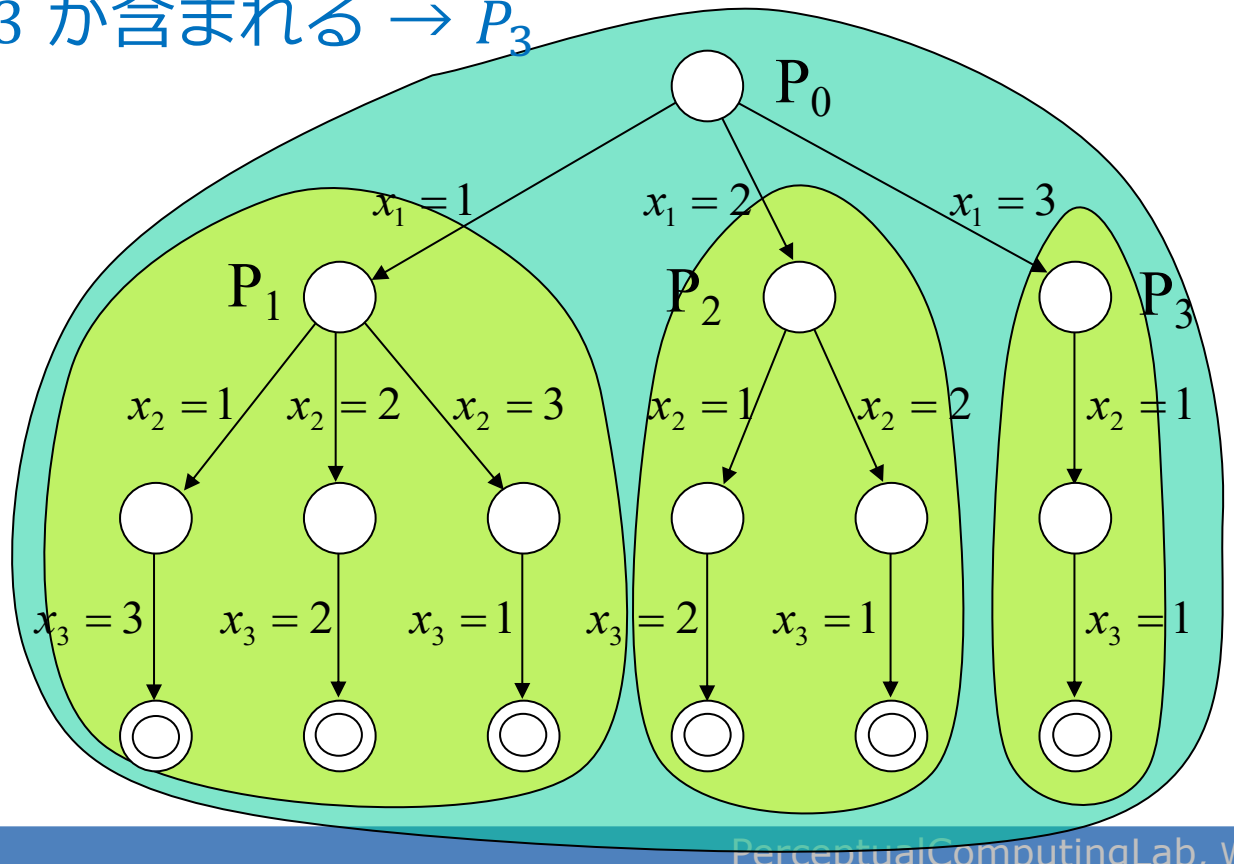
問題 P_0 からのびる3つの枝の意味：

$x_1 = 1$ の枝：　解に $x_1 = 1$ が含まれる → P_1

$x_1 = 2$ の枝：　解に $x_1 = 2$ が含まれる → P_2

$x_1 = 3$ の枝：　解に $x_1 = 3$ が含まれる → P_3

なる３つの部分問題に
問題を分解している。



分枝限定法の考え方

基本的な考え方：

- 与えられた問題 P_0 を複数の部分問題 P_i $(i = 1,2,\cdots)$ に分解する。（分枝操作）
- 部分問題 P_i を何らかの方法で終端する。（限定操作）

終端の条件：

- 探索中にその部分問題 P_i の最適解が求まった場合。
- その部分問題に 問題 P_0 の最適解が存在しないことがなんらかの方法で明らかになった場合。

限定操作

限定操作とは：

- 分解された部分問題に対し，それ以上調べる必要がないと判断して，探索を打ち切る処理。

限定操作：

- 下界値による限定操作
- 優越関係による限定操作

下界値による限定操作

緩和問題：

- ある部分問題 P_i に対し，その緩和問題 P'_i を以下のように定義する。

P_iの緩和問題 P'_i ：

目的関数： $g(x) \rightarrow$ 最小

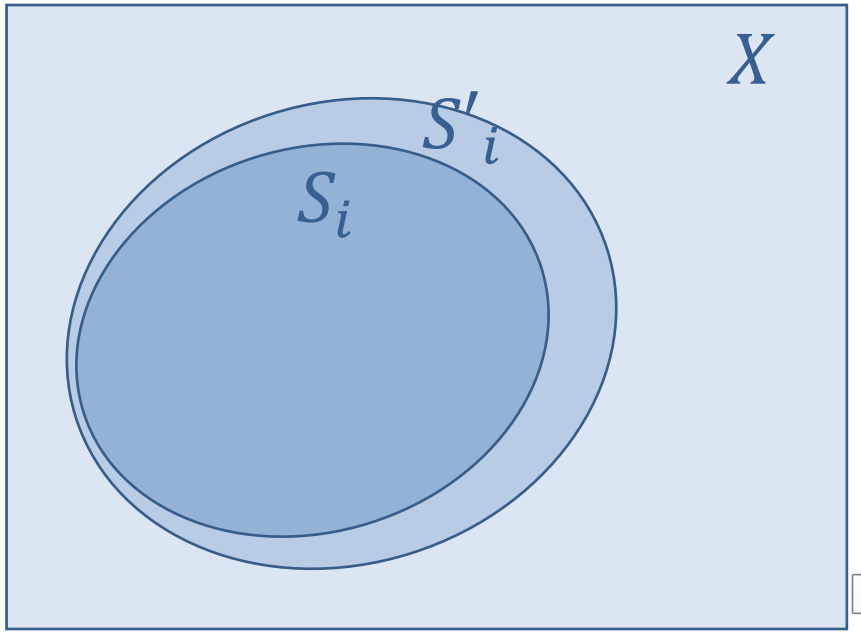
制約条件： $g(x) \leq f(x),\ x \in S_i$

ただし， $x \in S'_i,\ S_i \subset S'_i \subset X_i$

X_i は基礎となる空間，

S_i その実行可能領域（許容領域）

S'_i は，緩和問題の許容領域



下界値による限定操作

下界値：

- 部分問題 P_i の最適値を $f(P_i)$，$f(P_i) = \min_{x \in S_i} f(x)$

  その緩和問題 P'_i の最適値を $g(P_i)$，$g(P_i) = \min_{x \in S'_i} g(x)$

  とするとき，$g(P_i) \leq f(P_i)$ が成立する。$g(P_i)$ を下界値と呼ぶ。
  （$g(P_i)$ は $f(P_i)$ を楽観的に見積もったときの予想値）

下界値による限定操作

下界値：

- 部分問題 P_i の最適値を $f(P_i)$，$f(P_i) = \min_{x \in S_i} f(x)$

  その緩和問題 P'_i の最適値を $g(P_i)$，$g(P_i) = \min_{x \in S'_i} g(x)$

  とするとき，$g(P_i) \leq f(P_i)$ が成立する。$g(P_i)$ を下界値と呼ぶ。
  （$g(P_i)$ は $f(P_i)$ を楽観的に見積もったときの予想値）

緩和問題あるいは下界値の性質：

- P'_i が許容解を持たなければ，P_i も持たない。
- $g(P_i)$ (P'_iの最適値)が，$u(P_i)$ (許容解;$f(P_i)$の上界,$f(P_i) \leq u(P_i)$) に一致すれば，$f(P_i)$ (P'_iの最適値) に一致する。
- それまでに求められた元の問題 P_0 の許容解のうち最小値を z とするとき，$g(P_i) \geq z$ であれば，P_i およびそれより深い部分問題に解はない。

下界値による限定操作

緩和問題あるいは下界値の性質（言い換え）：

- P'_i が許容解を持たなければ，P_i も持たない。
  - 楽観的に見積もっても解がないなら，実際にも解はない。
- P'_i の最適値が P_i の許容解であれば，それは P_i の最適値でもある。
  - 楽観的に見積もった値を，実現する解があることが分かれば，その見積もり値が最適値である。
- それまでに求められた元の問題 P_0 の許容解のうち最小値を z とするとき，$g(P_i) \geq z$ であれば，P_i およびそれより深い部分問題に解はない。
  - 楽観的に見積もった値より優れた解が他にあるならば，その部分問題は最適値を与えない。

（ここで「楽観的に見積もる」とは，部分問題の最適値の予想値を，真の最適値より小さく与えることをいうものとする。）

優越関係による限定操作

- ある部分問題 P_i, P_j の間に，

$$f(P_i) \geq f(P_j)$$

なる関係があるとき，「 P_j は P_iに優越する」 という。

部分問題 P_i に優越する部分問題 P_j が既に生成されているのであれば，P_i を終端することができる。

分枝限定法の処理手順

＜記号の定義＞

A: 既に生成されているが，まだ分解も終端もしていない部分問題の集合。（アクティブリスト）

N: 既に生成されている部分問題全体の集合。

z:　暫定値。

O:　暫定値を与える許容解の集合。

s(A): Aから探索のためにひとつ部分問題を選ぶ操作。探索関数。

分枝限定法の処理手順

1. $A = \{P_0\}, N = \{P_0\}, z = \infty, O = \phi$
2. $A \neq \phi$ の間繰り返し
   - 2.1
     - $P_i = s(A)$
     - $u(P_i) < z$ $(x \in S_i, f(x) < z)$ であれば, $z = f(x), O = \{x\}$
     - P_iの緩和問題 P'_i が許容解を持たなければ break
     - $g(P_i) = u(P_i)$ であれば break
     - $g(P_i) \geq z$ であれば break
     - P_i より優越する $P_j \in N$ が既にあれば break
     - P_i の子節点 $P_{i1}, P_{i2}, P_{i3}, \cdots, P_{in}$ を作り

$$A = A \cup \{P_{i1}, P_{i2}, P_{i3}, \cdots, P_{in}\}$$
$$N = N \cup \{P_{i1}, P_{i2}, P_{i3}, \cdots, P_{in}\}$$

   - 2.2 $A = A - \{P_i\}$
3. 最適値はz , 最適解はOに記憶されたx

既に生成されているが，まだ分解も終端もしていない部分問題の集合。（アクティブリスト）

既に生成されている部分問題全体の集合。

Aから問題をひとつ選ぶ操作

上界値＝許容解の最適値 | 解 | 最適値 | 最適解

P_iの解が見つかった！

下界値よる限定

優越関係による限定

分枝操作＝部分問題を作る

探索の問題

探索関数 s(A) をどのように選ぶか

- 盲目的探索
  - 深さ優先探索　(Depth first search)
    - 深さの深いノードを優先的に探索する
  - 幅優先探索 (Breadth first search)
    - 深さの浅いノードを優先的に探索する

- ヒューリスティックな探索
  - 最良下界探索(Best-bound search)
    - 下界値の最も良いノードを優先的に探索する
    - 例： A* search

A*サーチ

g(n)：S(スタート)からノードnに至るコストの予想値。

一般には，探索の途中既に見つかっている n に至る実コスト。

h(n)：ノードnからG(ゴール)に至るコストの予想値。

実際のコストより小さめに見積もる(*)。

とするとるき，

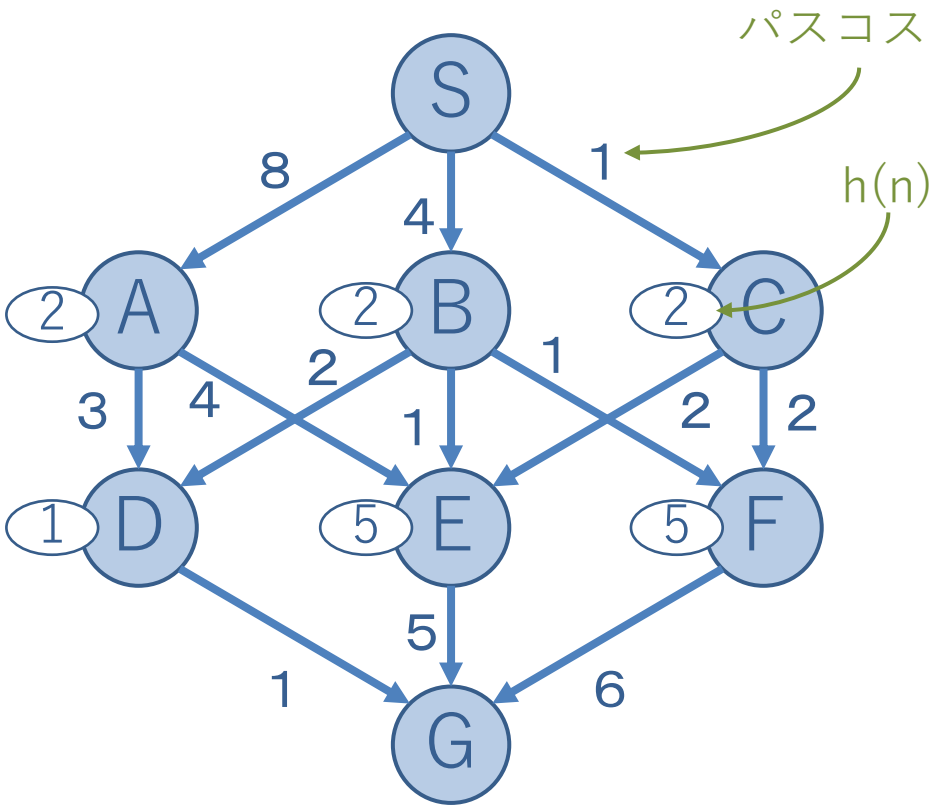
f(n)：Sからノードn経由でGに至るコストの予想値

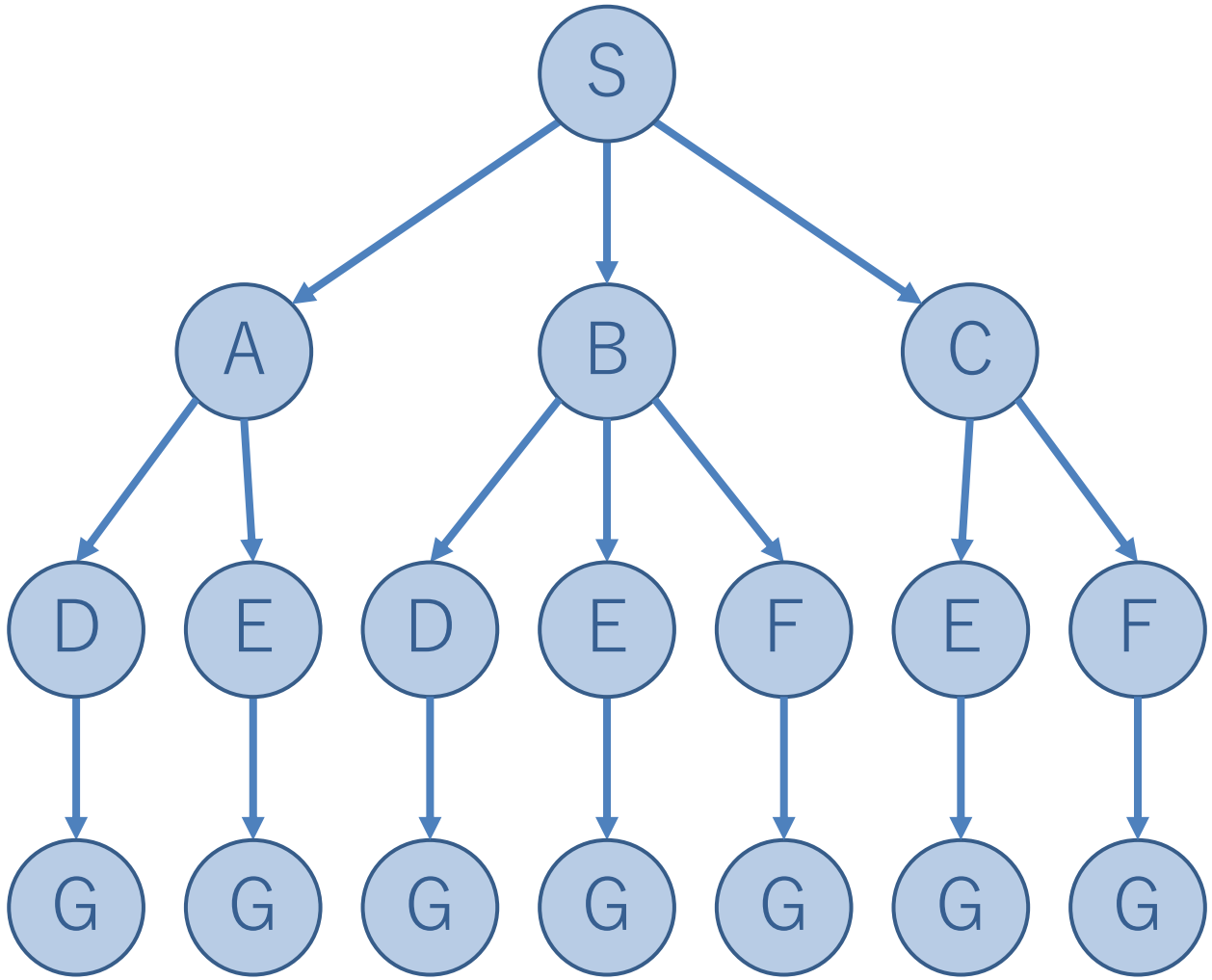
f(n)=g(n)+h(n)

f(n) は，h(n)が上記(*) の条件を満たすとき，下界値 としての性質を持つ。

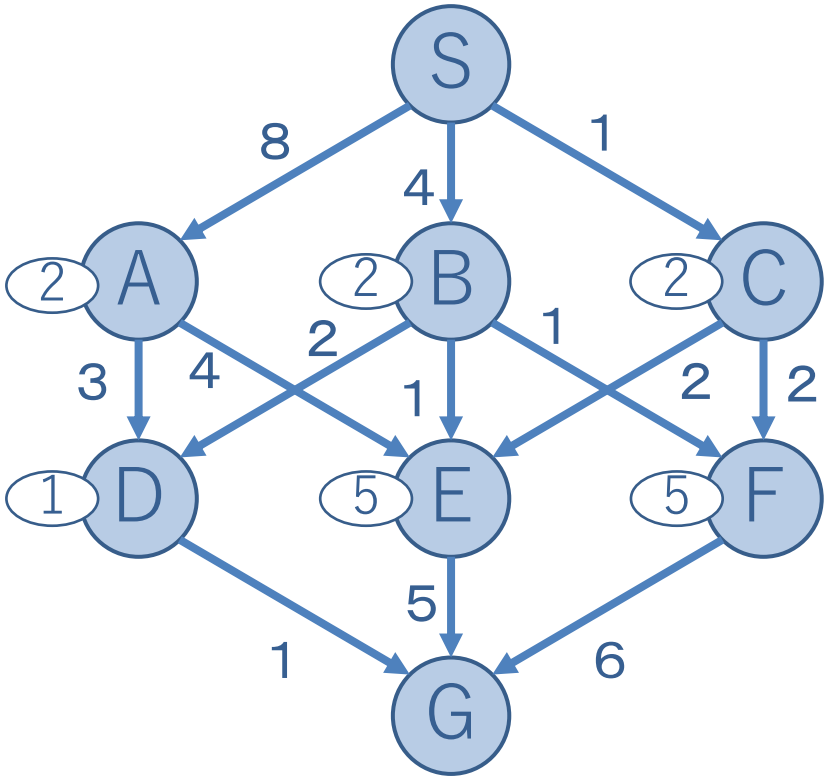
を探索のための評価関数（探索関数）として展開のためのノードを選ぶ探索手法。

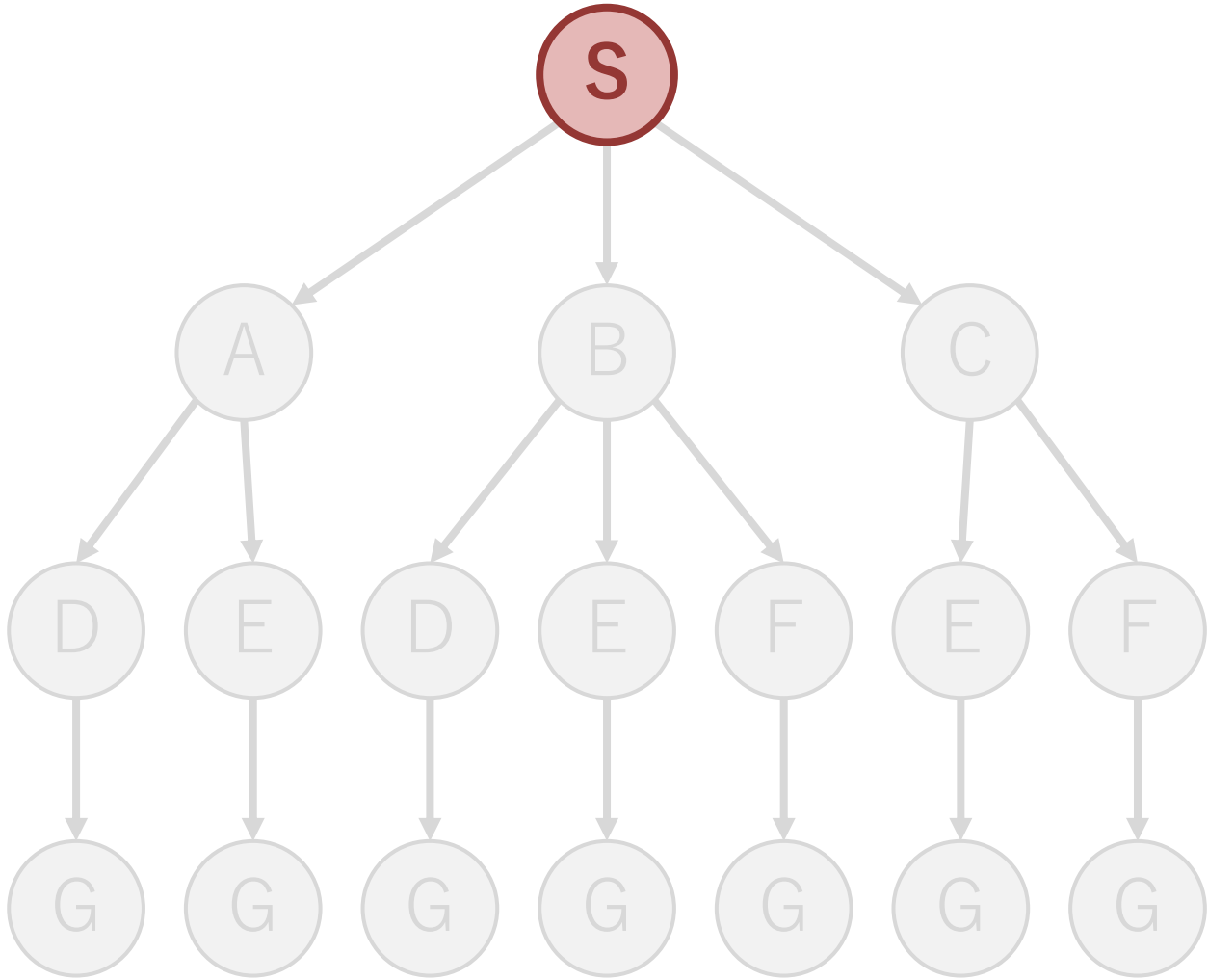
A*サーチ：探索木


パスコスト
h(n)
S
A
B
C
D
E
F
G
8
4
1
2
2
2
3
4
2
1
1
2
2
5
5
1
6


S
A
B
C
D
E
D
E
F
E
F
G
G
G
G
G
G
G

A*サーチ：探索木





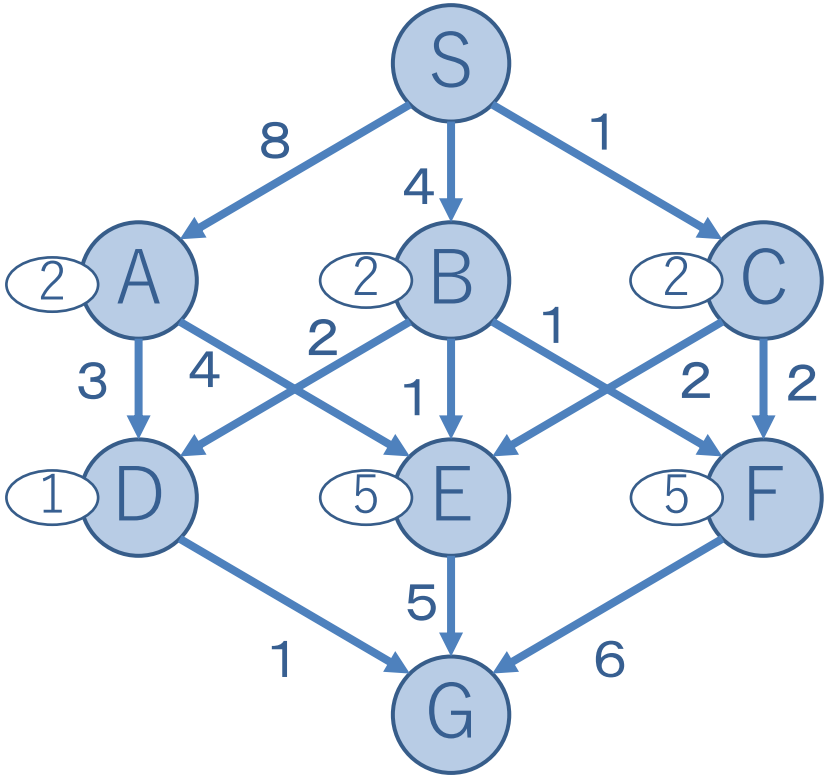
アクティブリスト中のノード

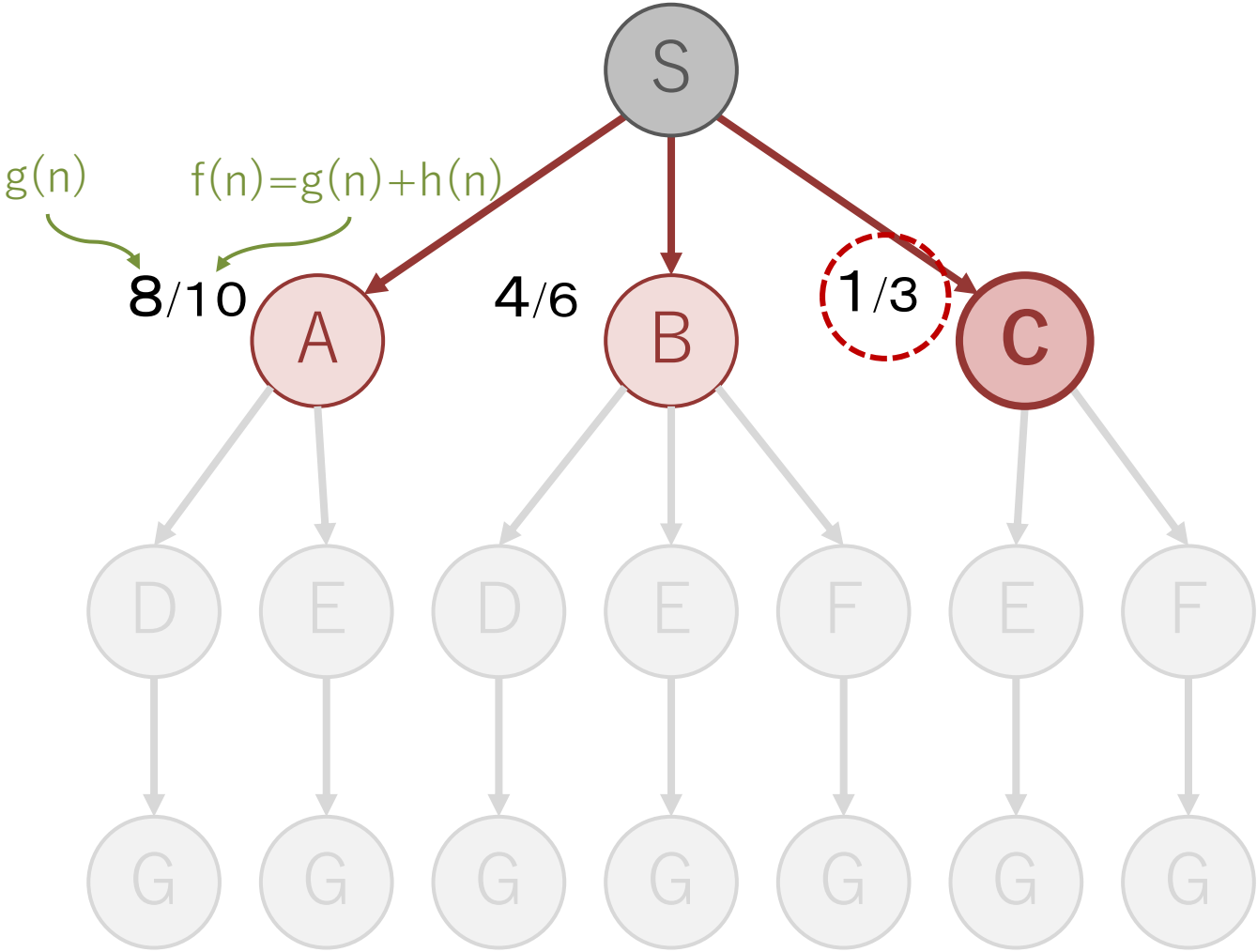
展開のために選ばれたノード

展開済みのノード

アクティブになっていないノード

A*サーチ：探索木



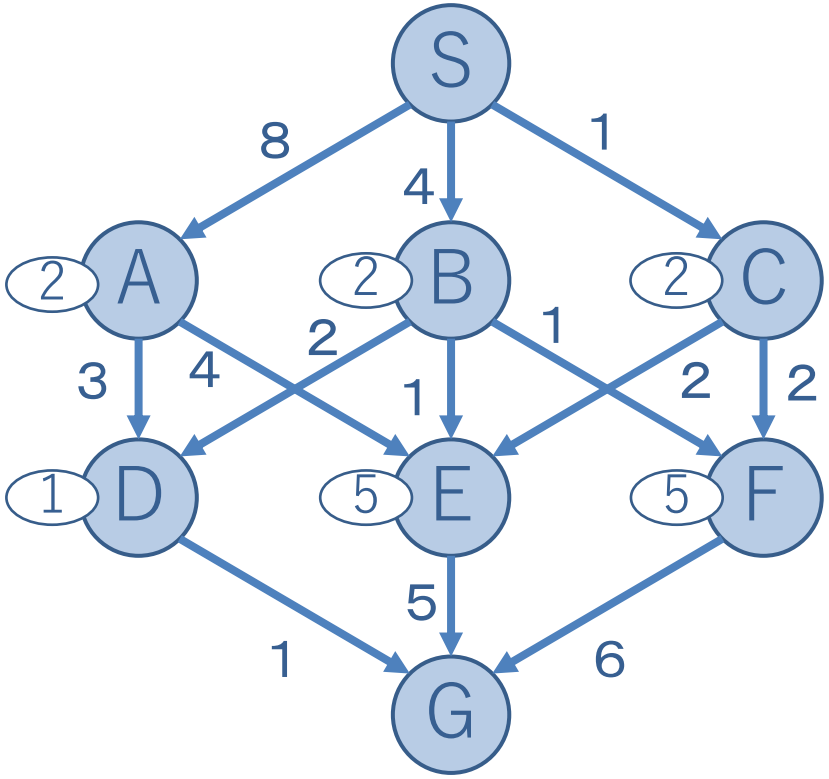


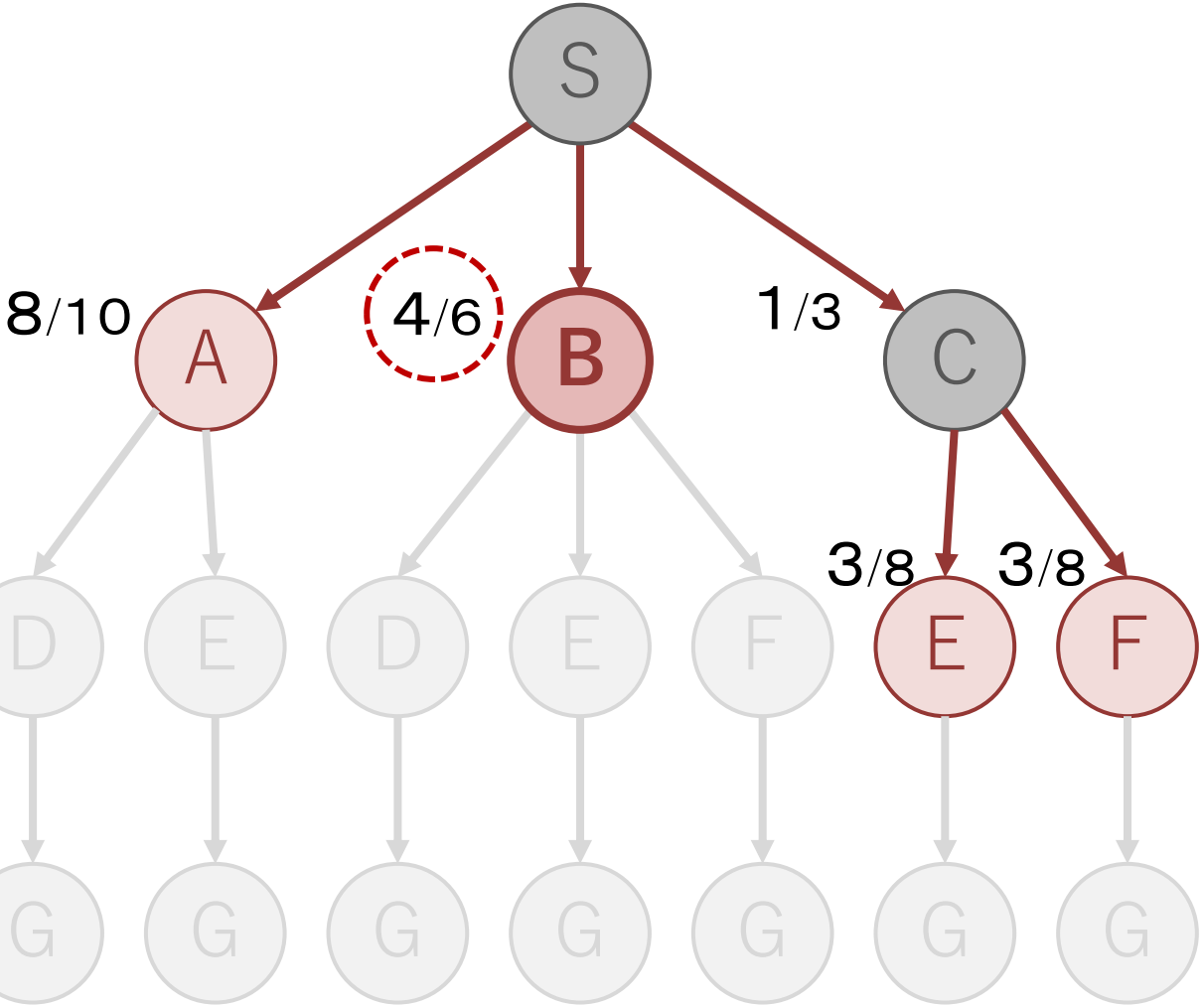
アクティブリスト中のノード

展開のために選ばれたノード

展開済みのノード

A*サーチ：探索木



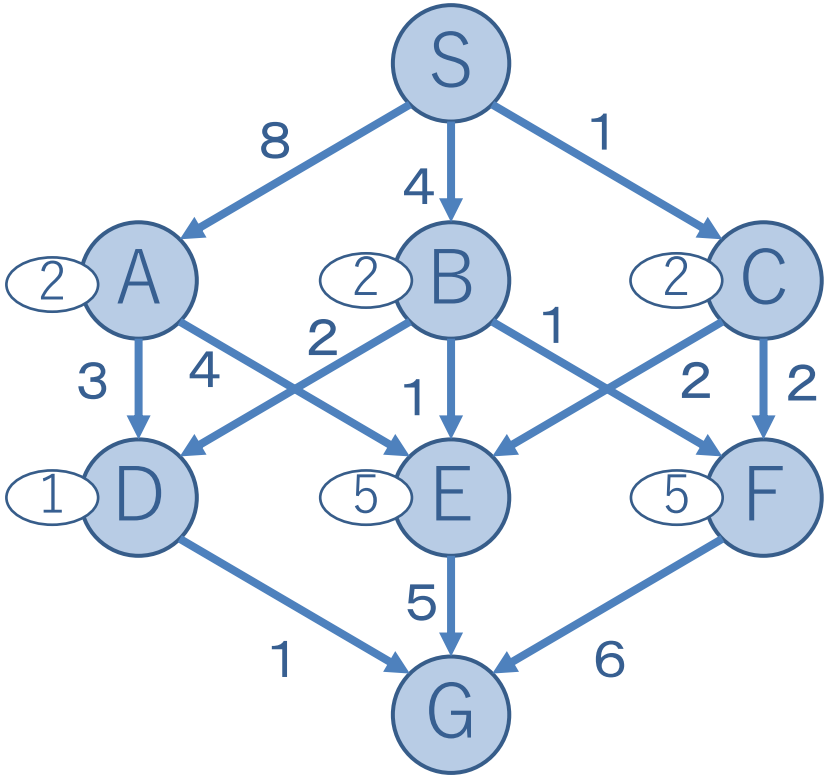


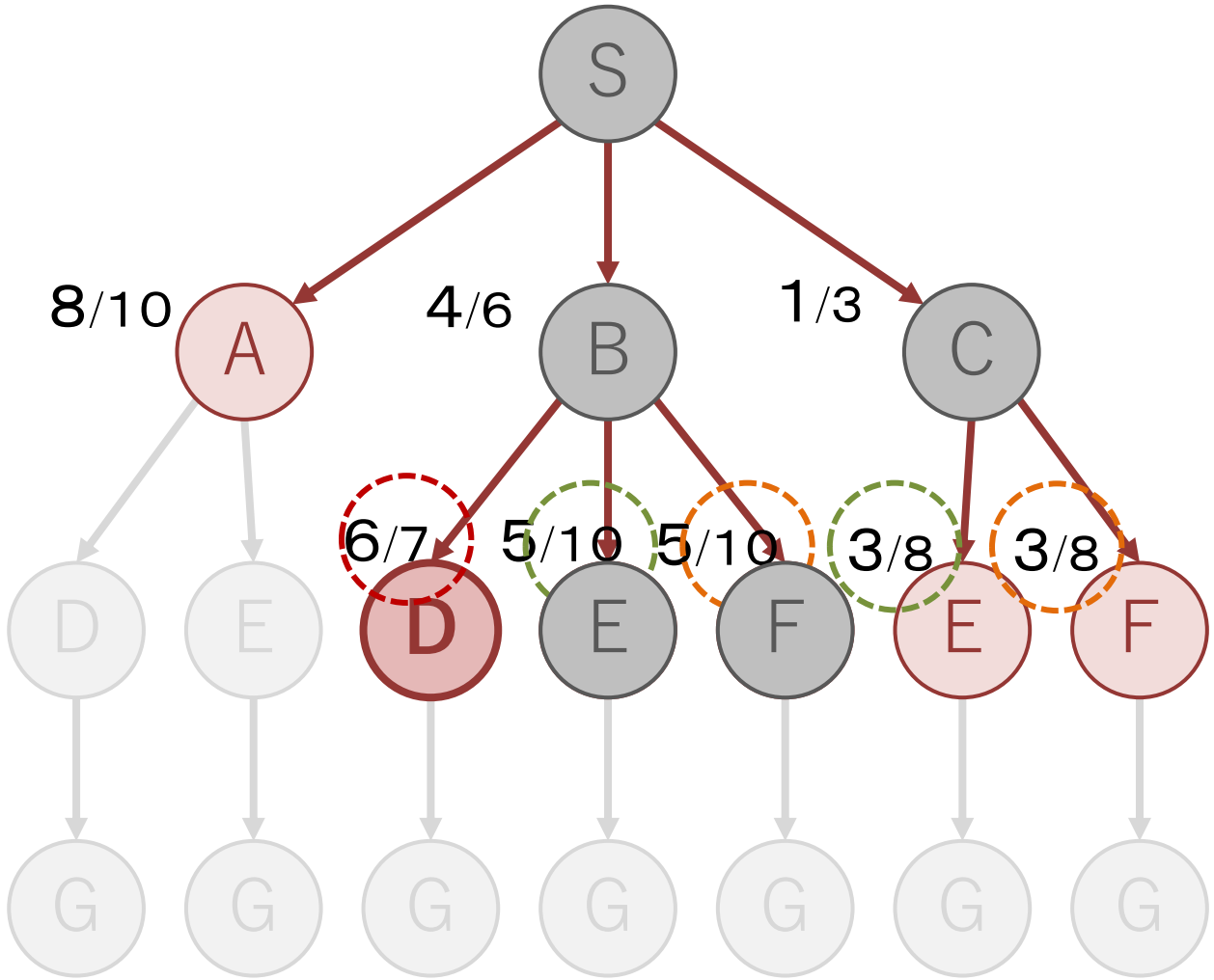
アクティブリスト中のノード

展開のために選ばれたノード

展開済みのノード

A*サーチ：探索木



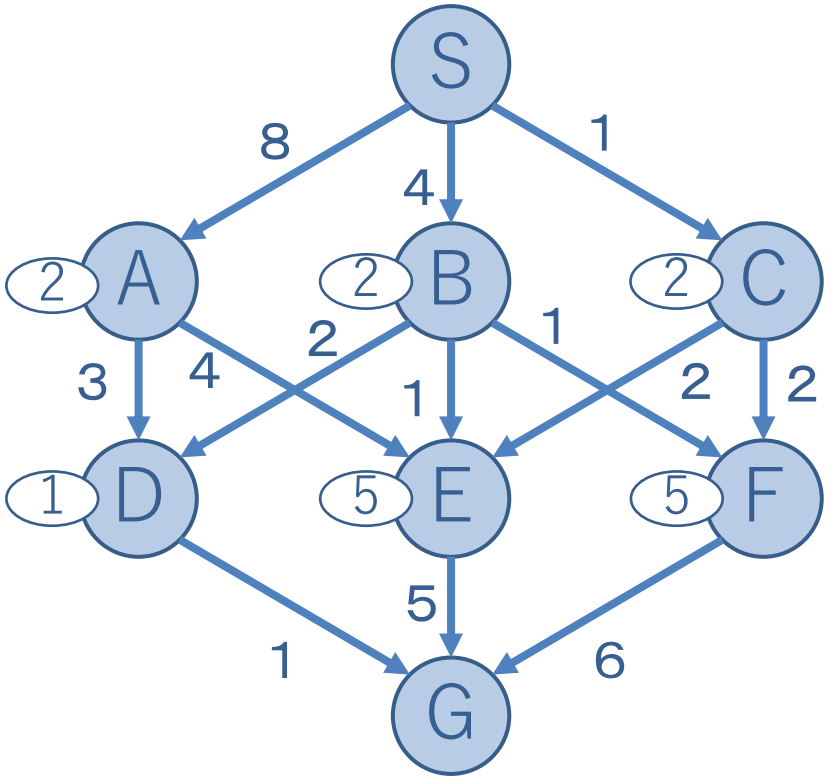


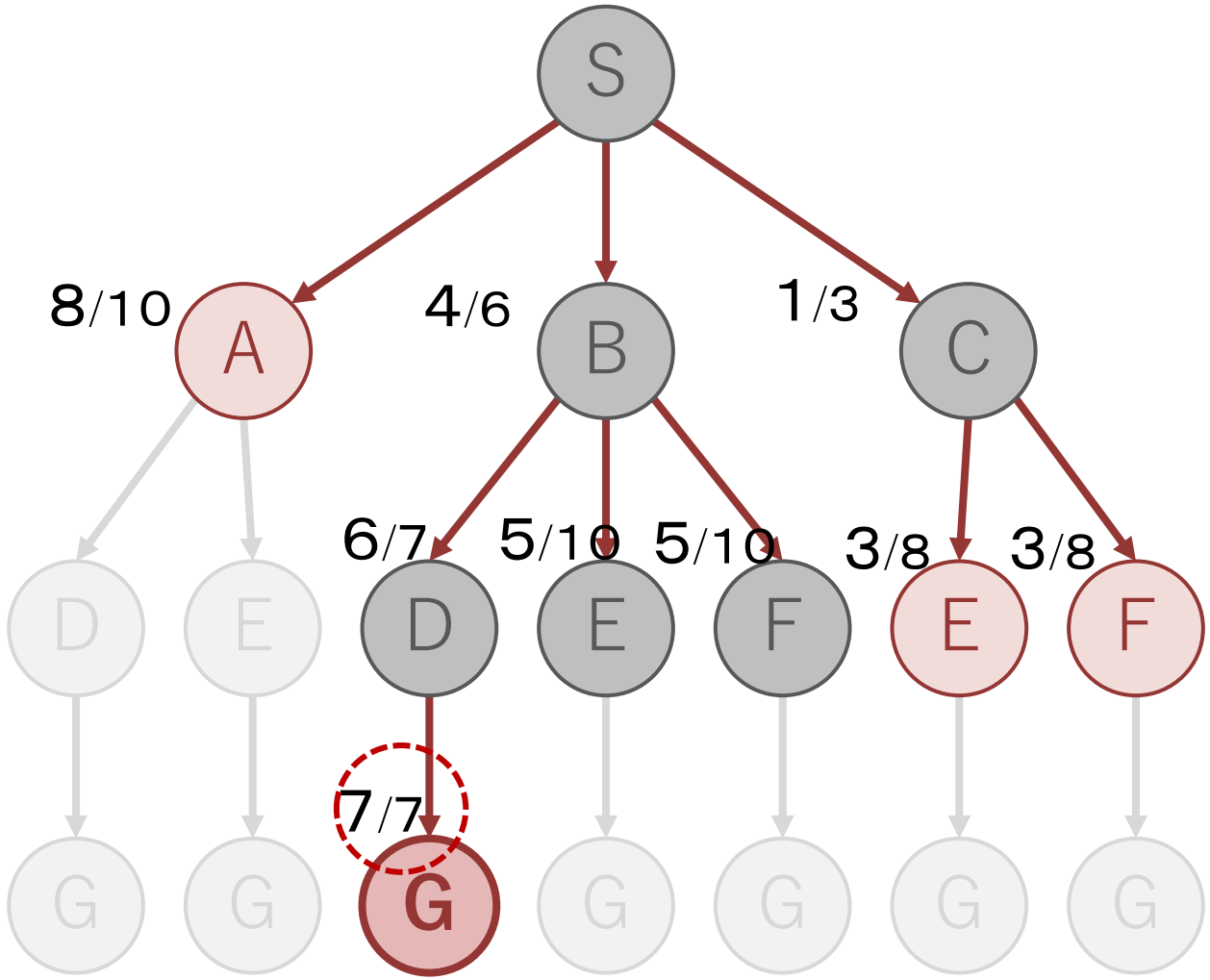
アクティブリスト中のノード

展開のために選ばれたノード

展開済みのノード

A*サーチ：探索木



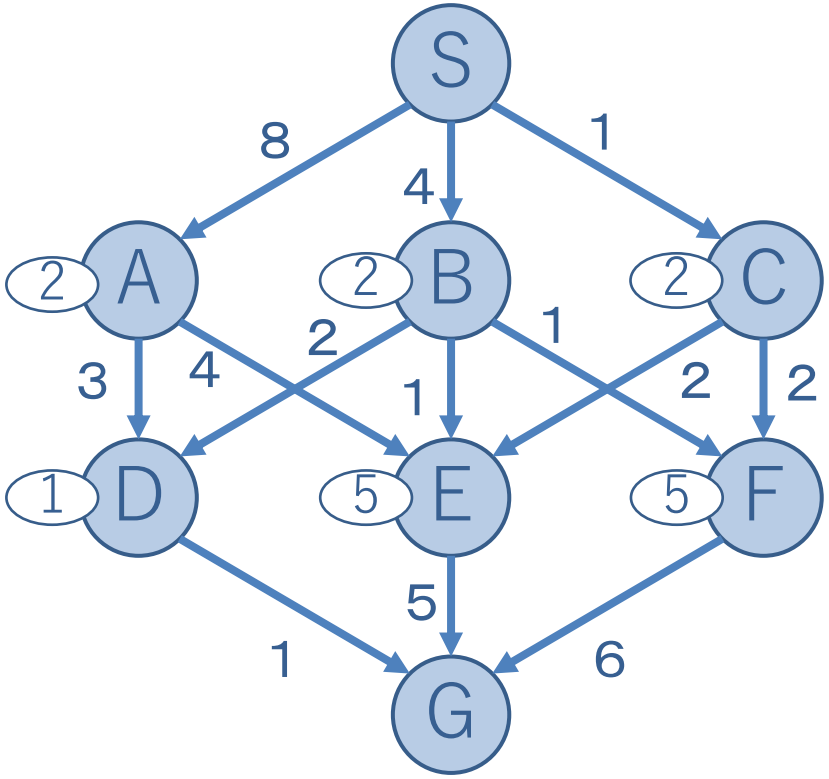


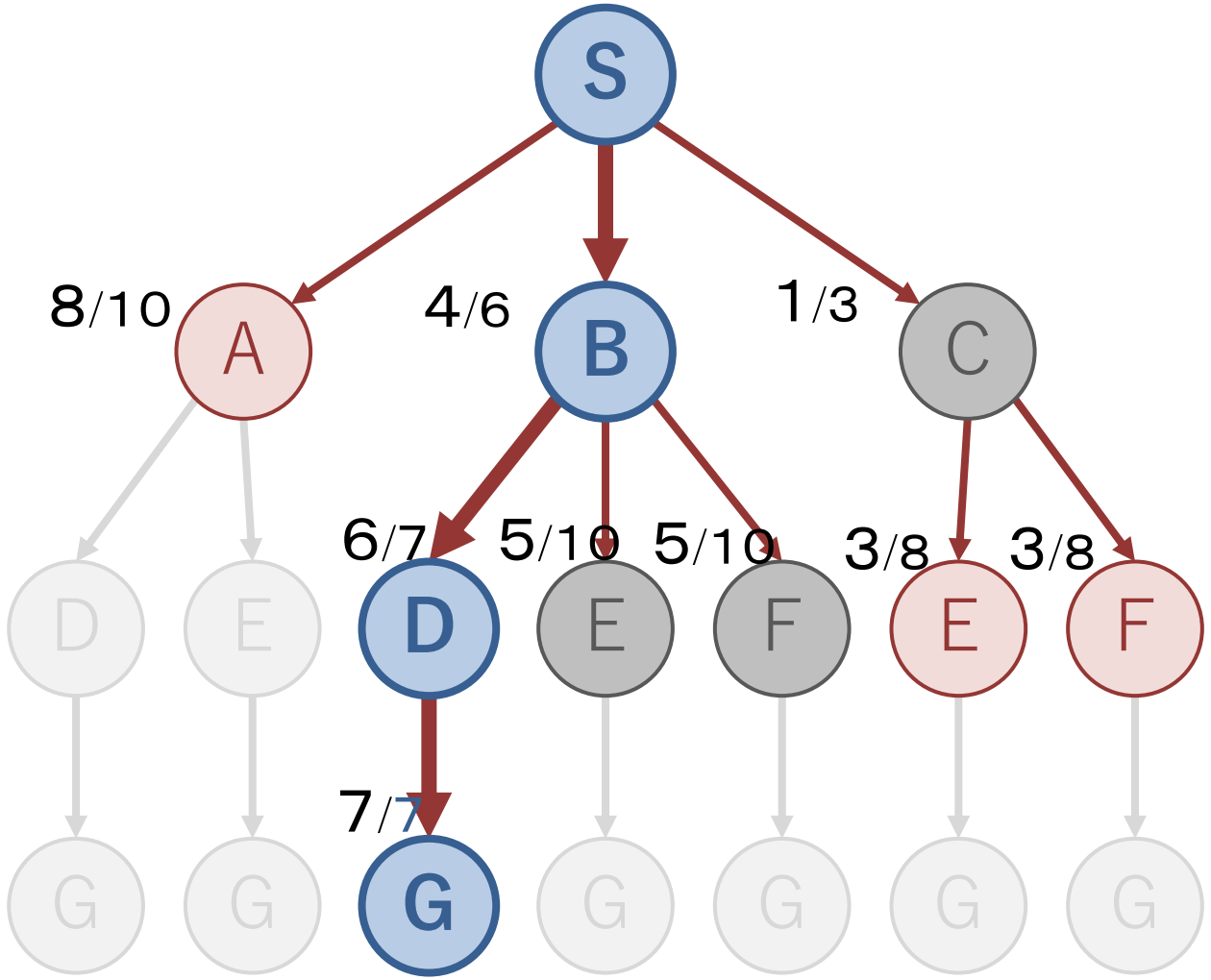
アクティブリスト中のノード

展開のために選ばれたノード

展開済みのノード

A*サーチ：探索木





アクティブリスト中のノード

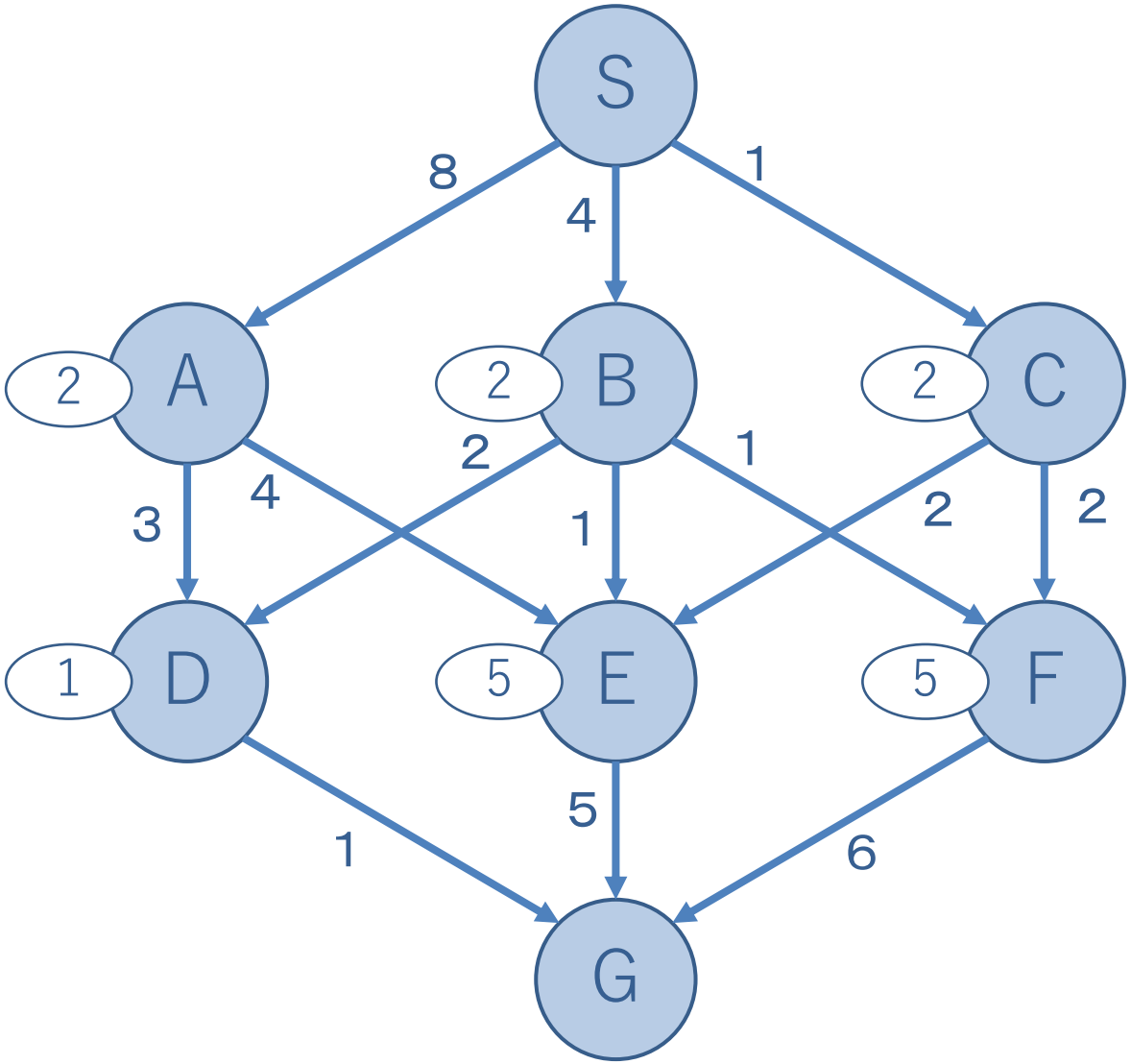
展開のために選ばれたノード

展開済みのノード

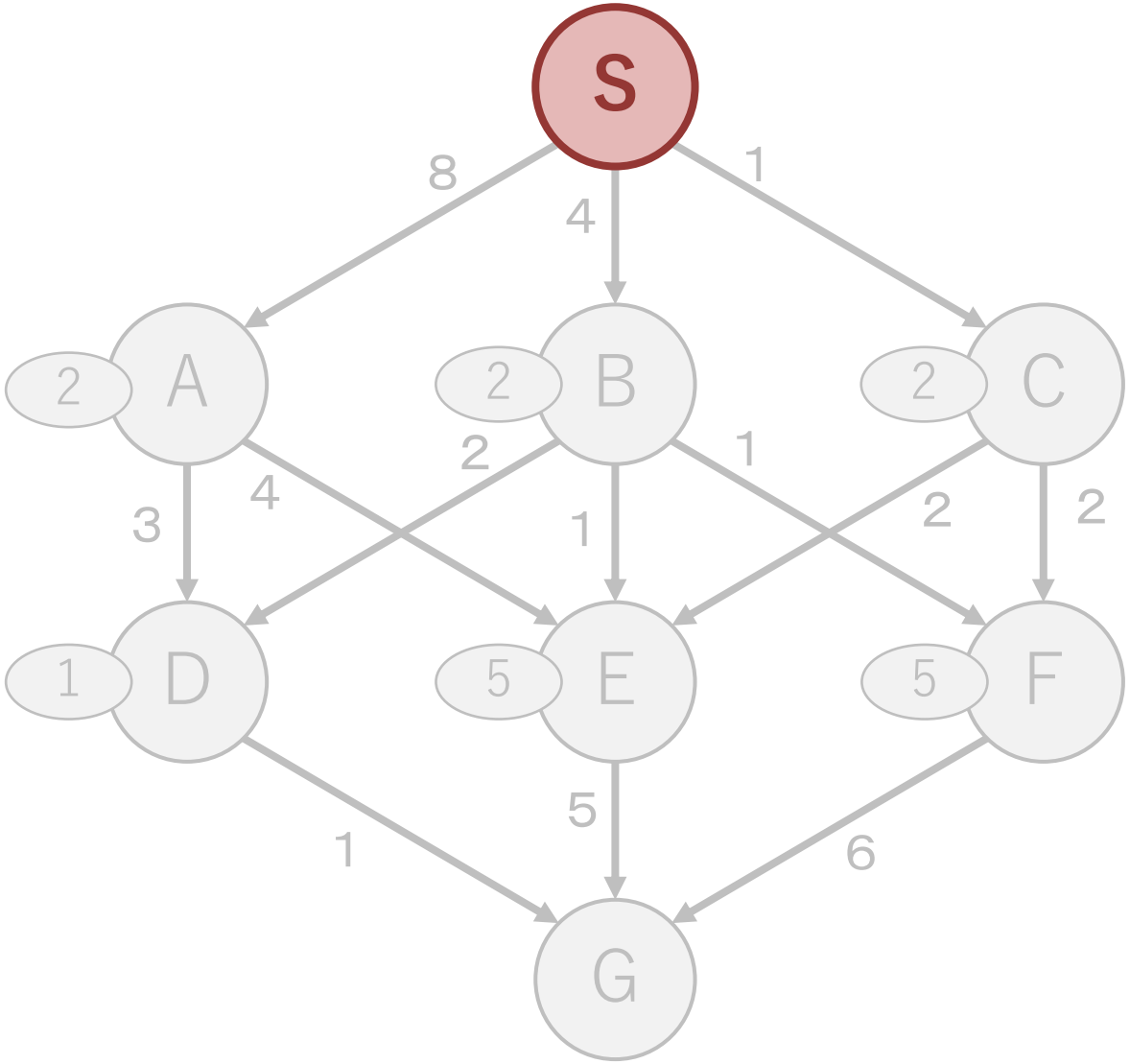
A*サーチ：グラフサーチ

1. アクティブリスト：$A = \{S\}$；クローズドリスト： $C = \phi$
2. $A \neq \phi$ の間繰り返し
   - Aの要素のうち，評価値 f = g + h の最も良いノードを n とする。
   - n を展開する（nから後継ノードへポインタを張り，n の後継ノードをアク ティブリスト A に入れる）
   - nの後継ノードが既にAにある場合，gの値の悪い方のポインタをはずす。
   - nの後継ノードが既にCにある場合，gの値の悪い方のポインタをはずすとともに，gの値の差，fの値の差を，子孫ノードに伝搬する。
   - n を クローズドリスト C に入れる
3. リンクをバックトラックして最適経路を得る。

A*サーチ：グラフサーチ

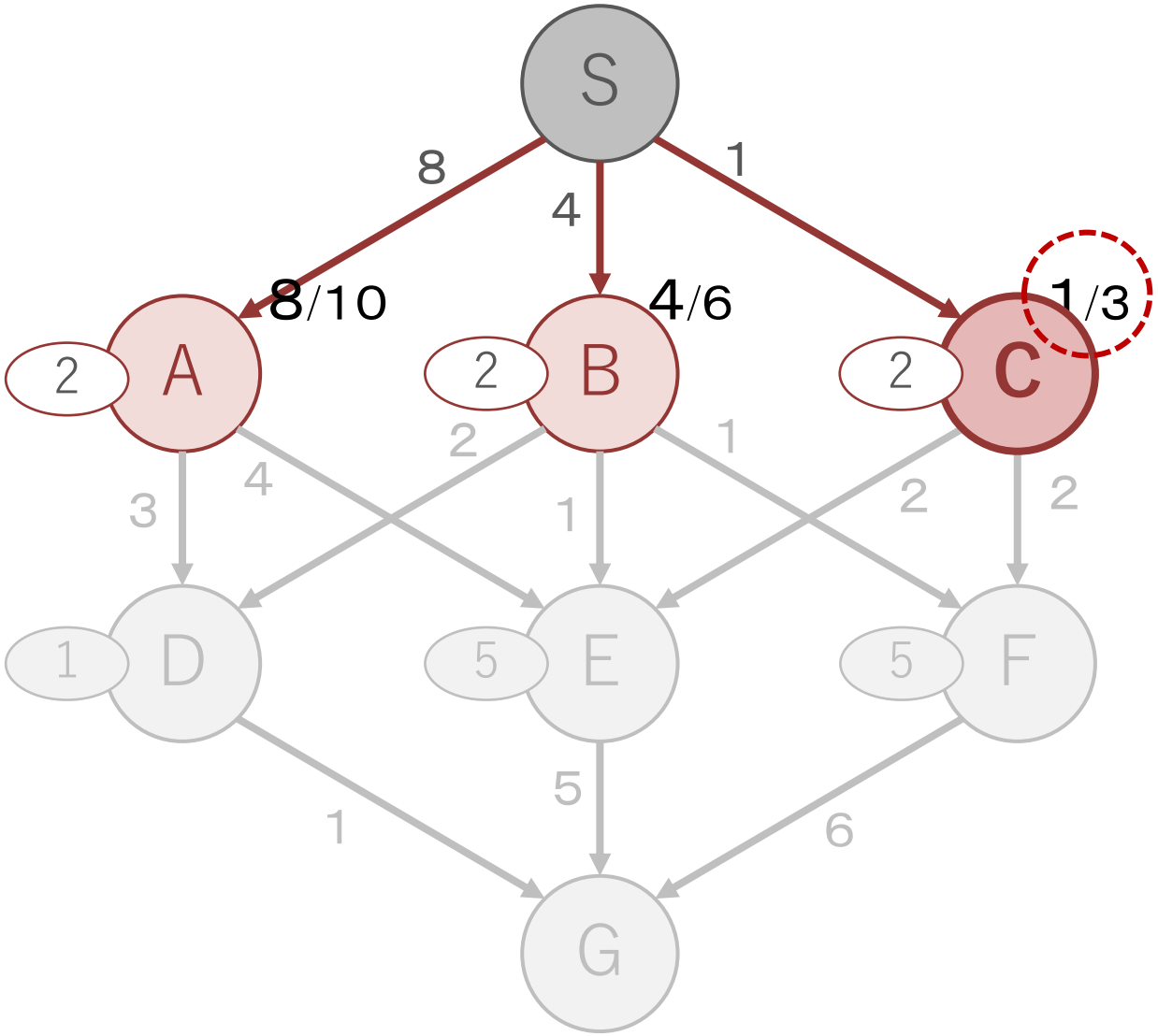


A*サーチ：グラフサーチ



アクティブリスト中の
最良評価値を持つノードを選ぶ

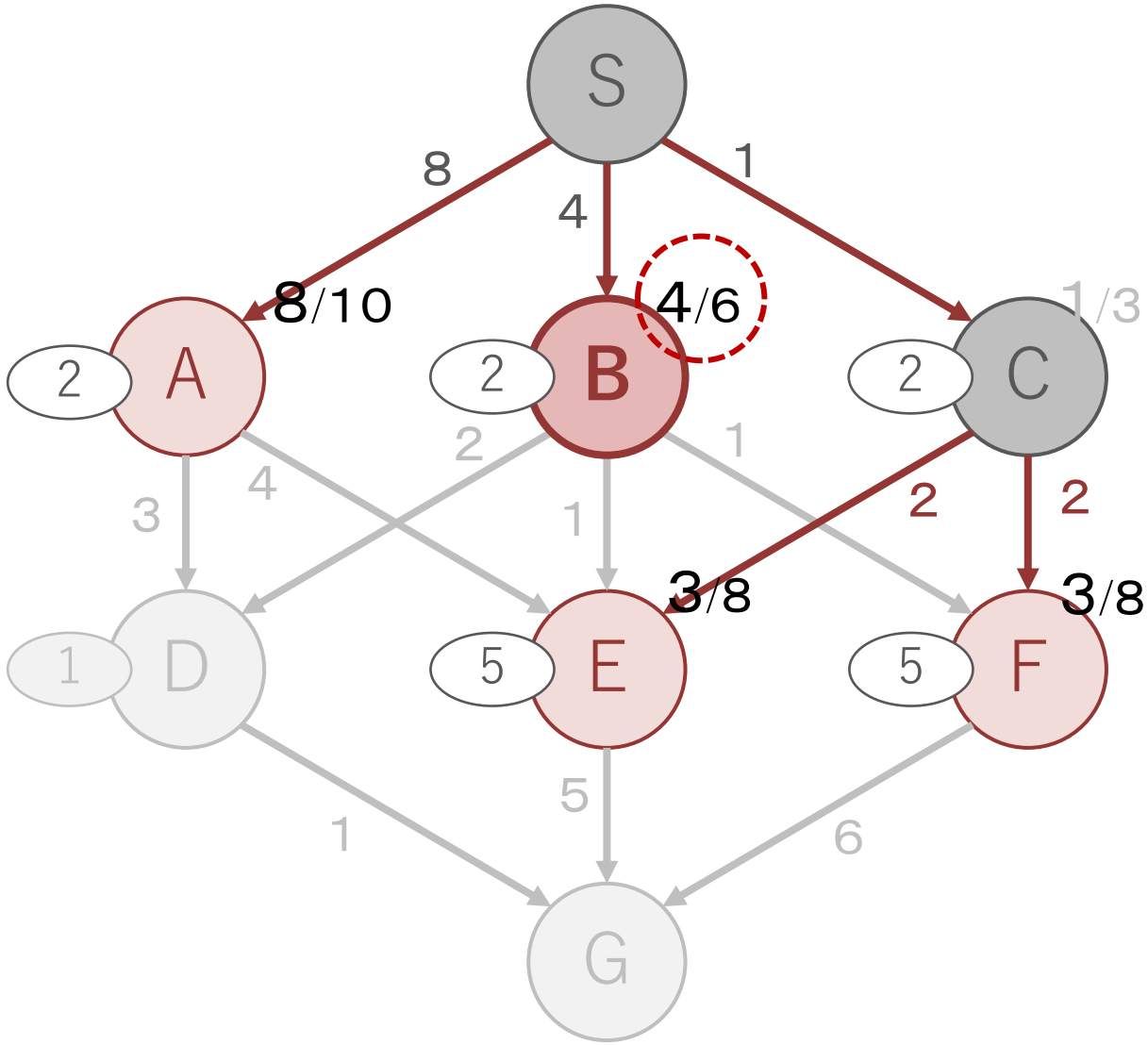
A*サーチ：グラフサーチ



アクティブリスト中の
最良評価値を持つノードを選ぶ

選んだノードを展開する

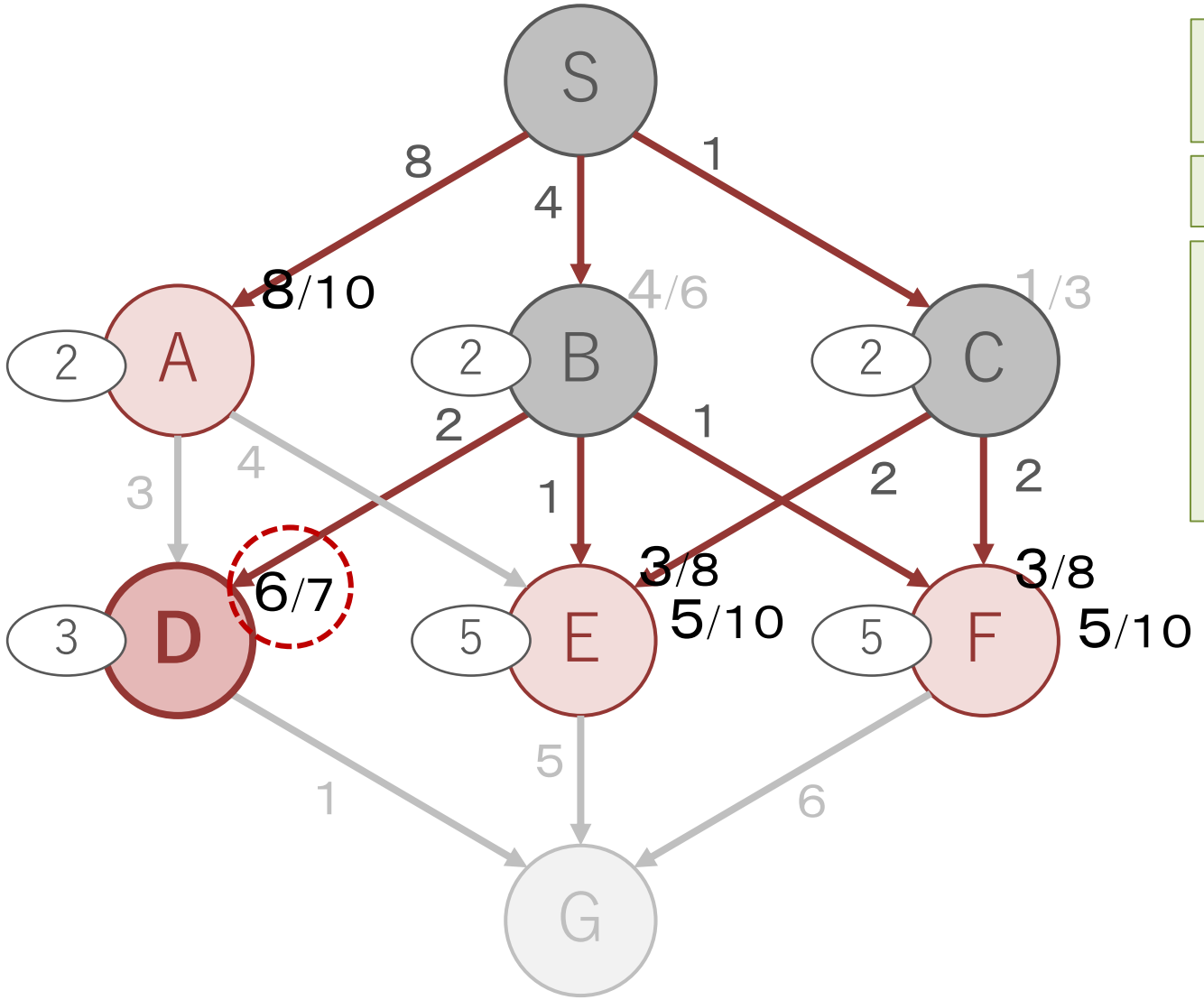
A*サーチ：グラフサーチ



アクティブリスト中の
最良評価値を持つノードを選ぶ

選んだノードを展開する

A*サーチ：グラフサーチ

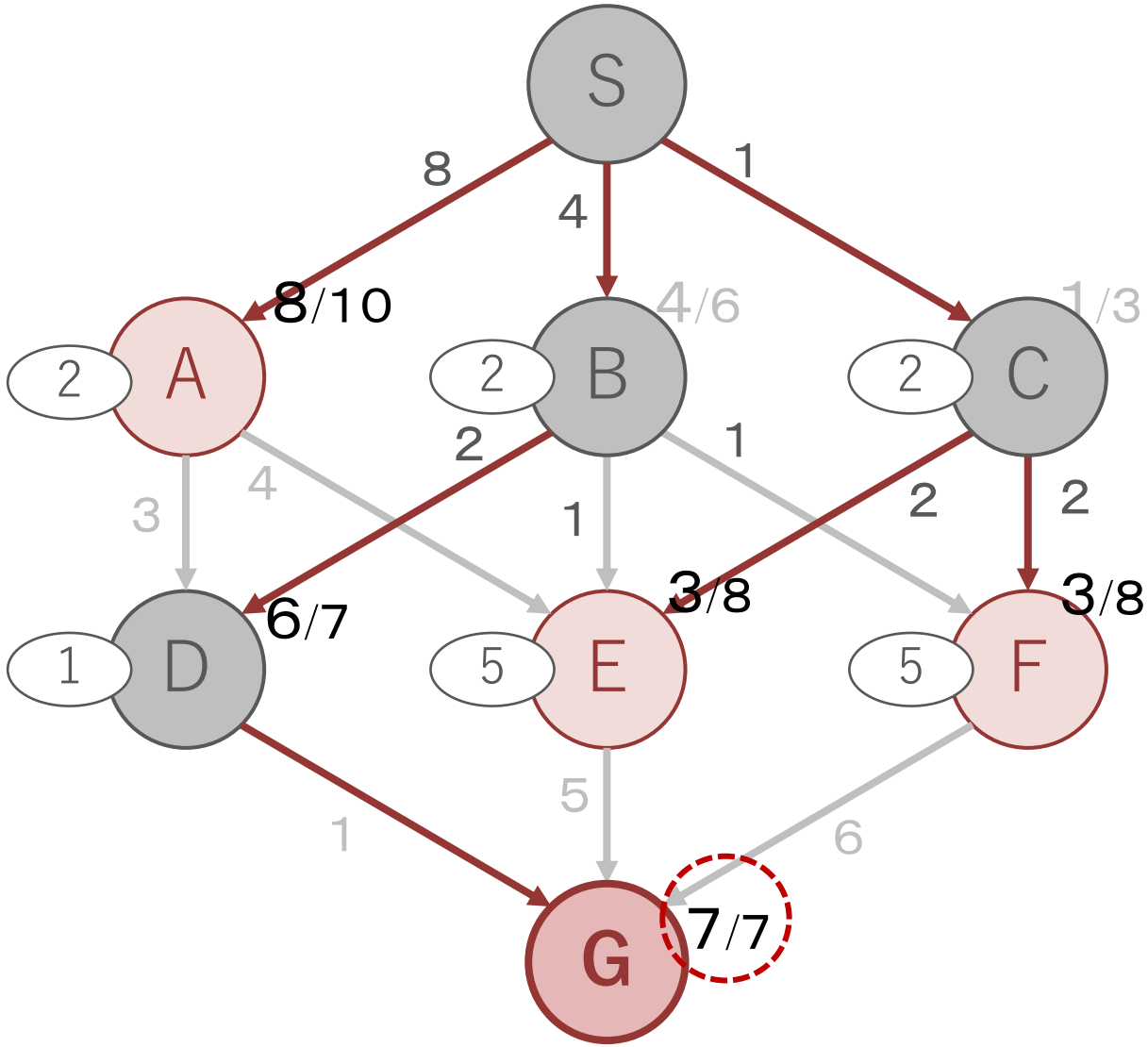


アクティブリスト中の
最良評価値を持つノードを選ぶ

選んだノードを展開する

既にアクティブリストにあるノードは，既発見のルートとどちらが良いかを調べ，良い方のルートを残す（悪い方からのポインタをはずす）

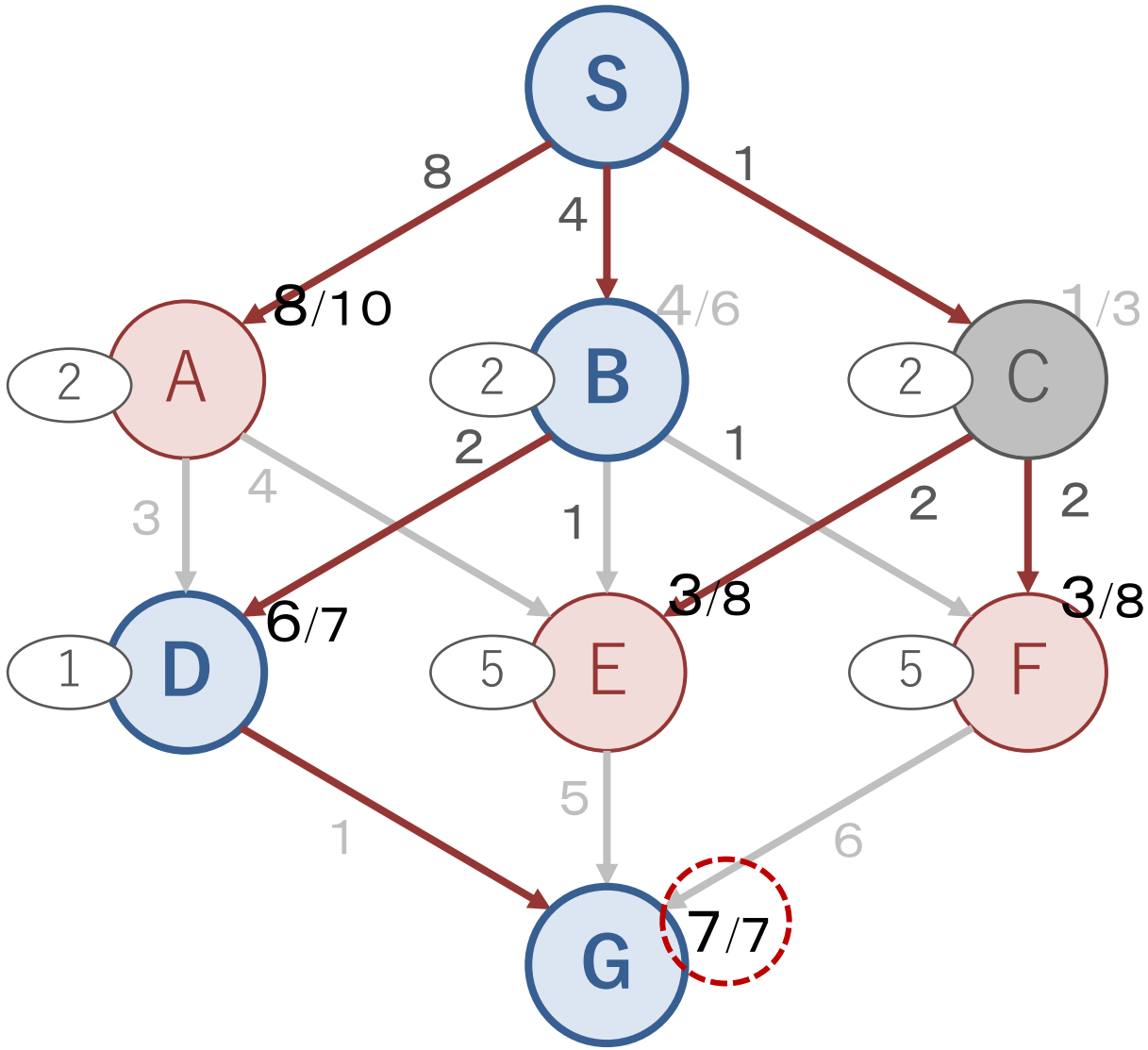
A*サーチ：グラフサーチ



アクティブリスト中の
最良評価値を持つノードを選ぶ

選んだノードを展開する

A*サーチ：グラフサーチ



アクティブリスト中の
最良評価値を持つノードを選ぶ

ゴールノードが選ばれたら，終了。バックトラックして最適経路を出力。

最適性の証明

A*探索は，必ず最適解を見つけて終了する。

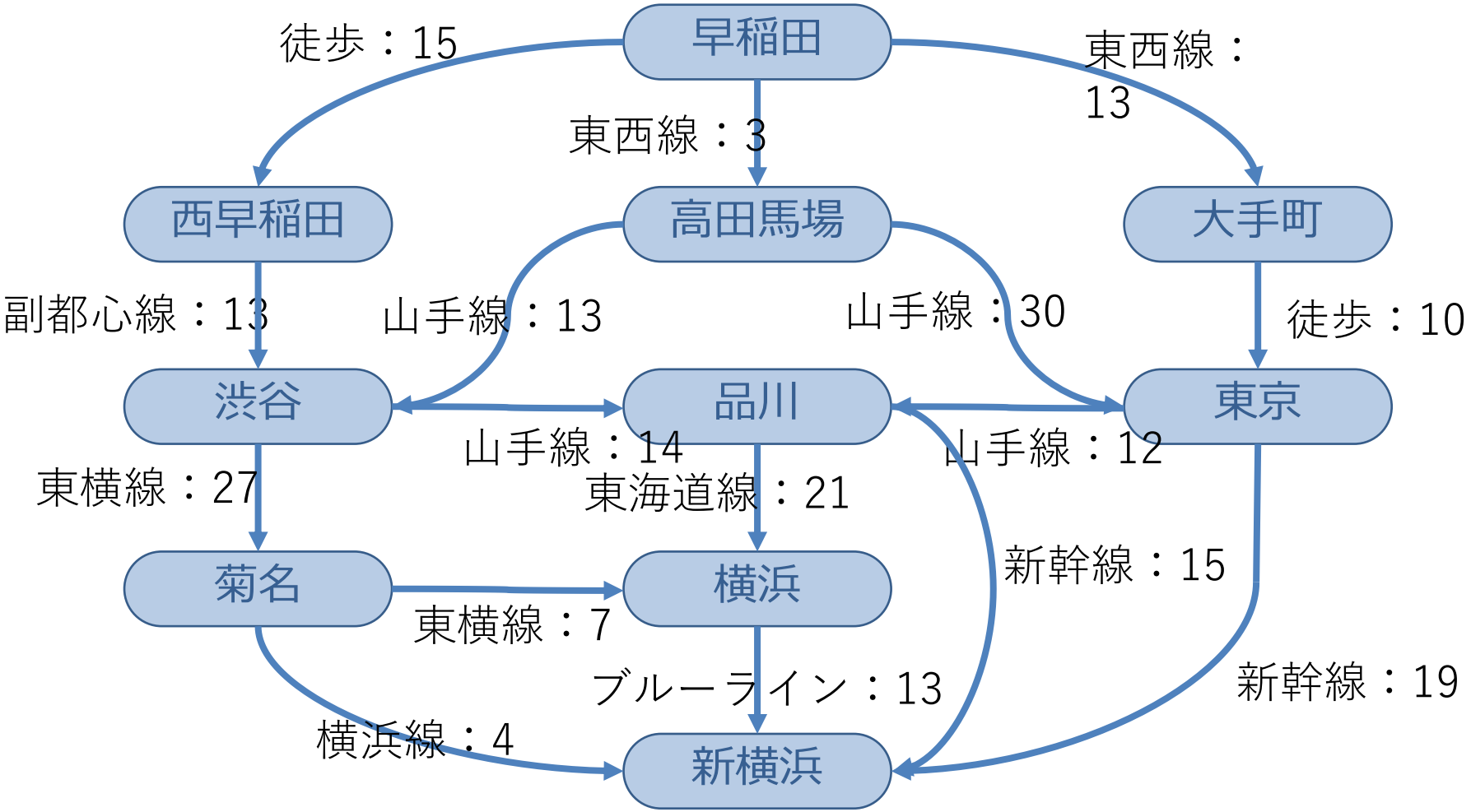
1) 最適パスを O_0（=S) O_1 O_2 ．．．O_i O_{i+1} ．．．O_N（=G)
とするとき，探索の途中において，アクティブリストには必ず最適パス上のノードが存在する。
   - 1-1) 初期状態において 最適パス上のノードO_0 が存在する。
   - 1-2) O_iがアクティブリストにあって，これが展開されアクティブリストから外れるとき，必ずO_{i+1}がアクティブリストに入る。
   - 1-3) O_Nがアクティブリストから外れるとき，探索は終了する。
   - 1-4) 1-1)－1-3)より，1) が成立する。
2) 最適パス上のノードは必ず展開される
   - 2-1) アクティブリストにある最適パス上のノード O_i に対し，

     $$f(O_i) \leq f(G)$$

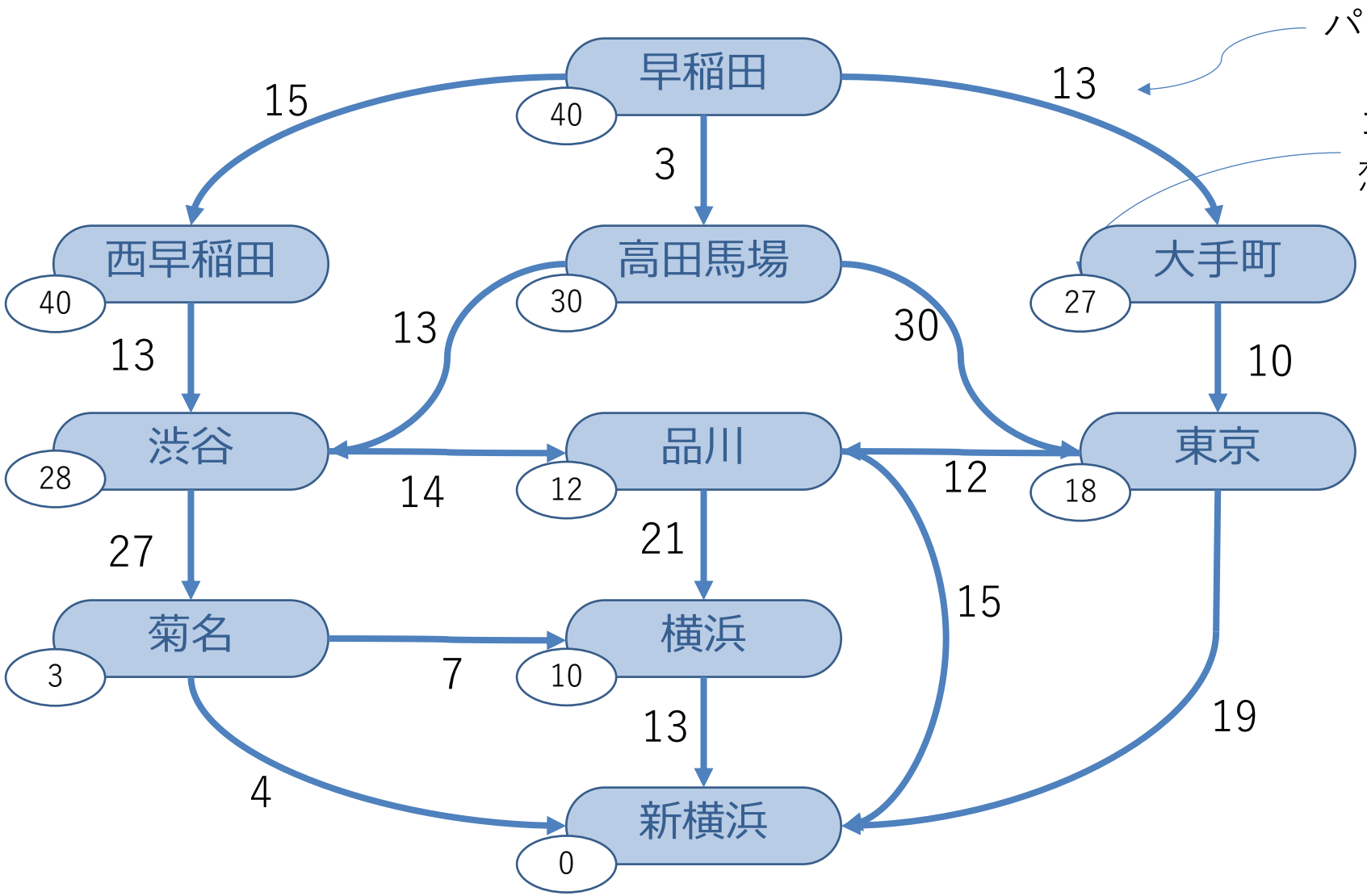
     終了条件は，f(G)が最小になること（Gが展開のために選ばれること）であるから，必ず O_i は展開のために選ばれる。
3) 1),2) より，Gは必ず展開される＝最適解が見つかる。

例題

- 早稲田から新横浜まで，最短時間で行く経路を求める。



h(n) (ゴールに至るコストの予想値）を用いる場合



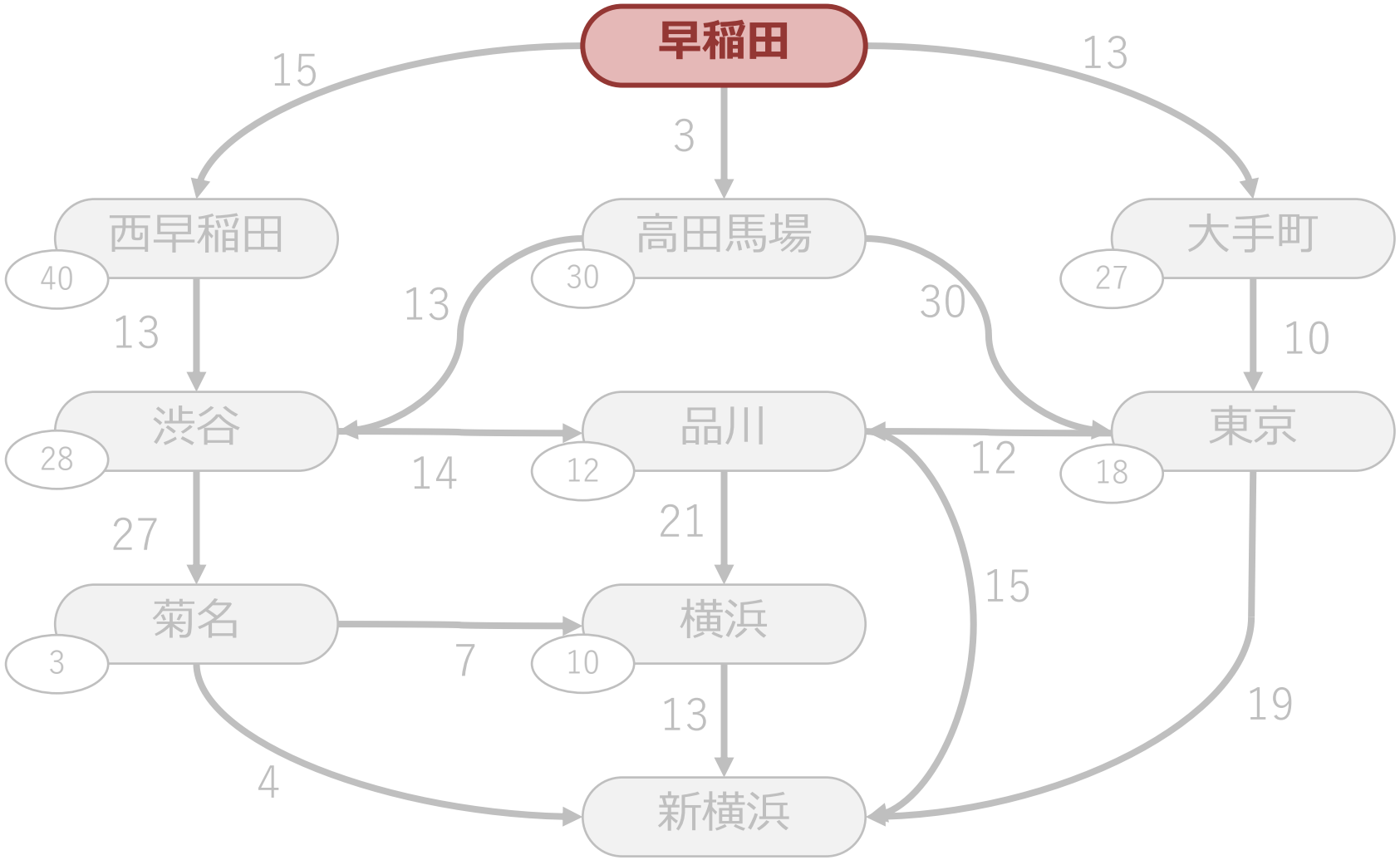
早稲田 アクティブリスト中のノード

早稲田 展開のために選ばれたノード

早稲田 展開済みのノード

早稲田 アクティブになっていないノード

h(n) (ゴールに至るコストの予想値）を用いる場合



アクティブリスト中の
最良評価値を持つノードを選ぶ

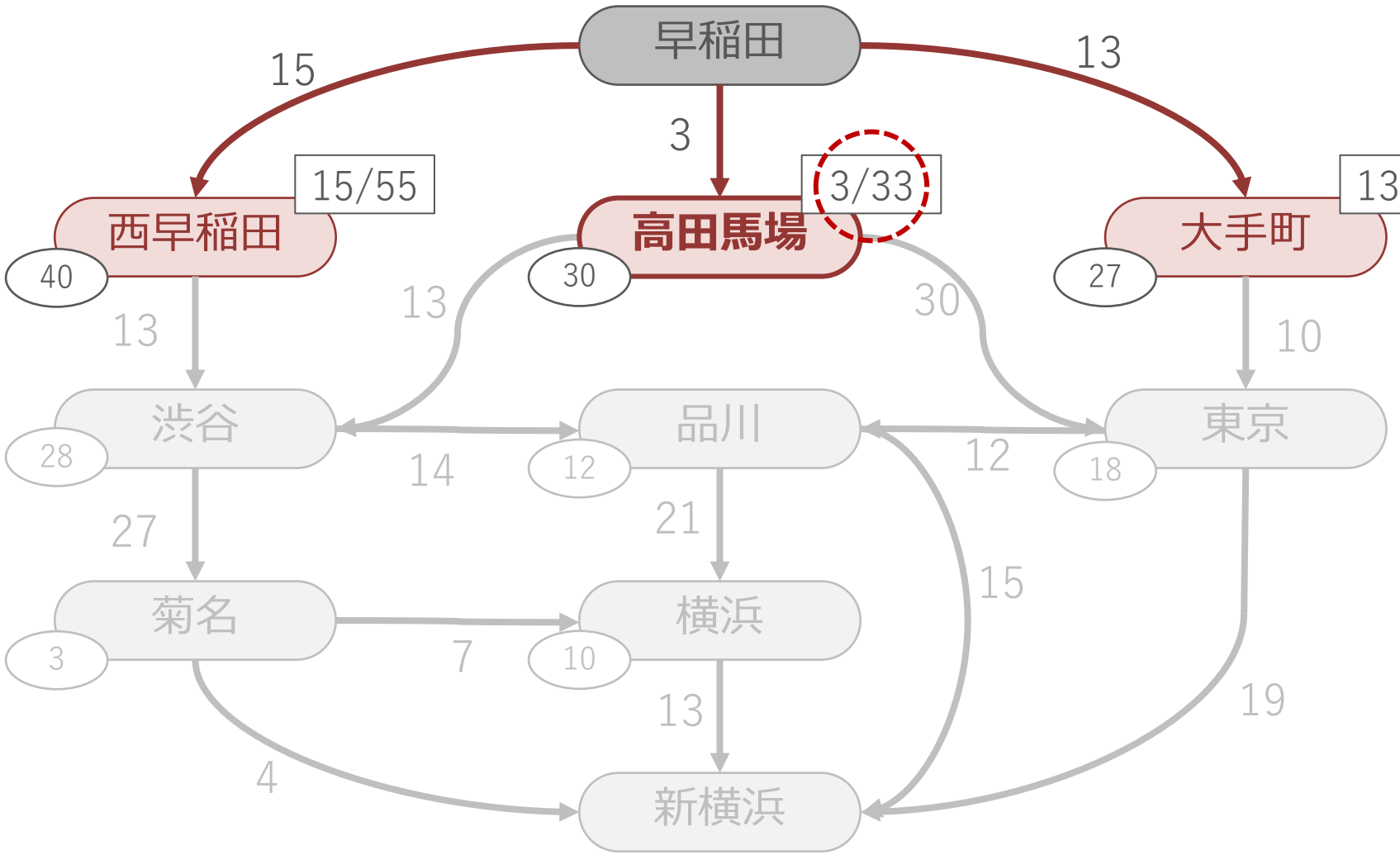
早稲田 アクティブリスト中のノード

早稲田 展開のために選ばれたノード

早稲田 展開済みのノード

早稲田 アクティブになっていないノード

h(n) (ゴールに至るコストの予想値）を用いる場合



アクティブリスト中の
最良評価値を持つノードを選ぶ

選んだノードを展開する

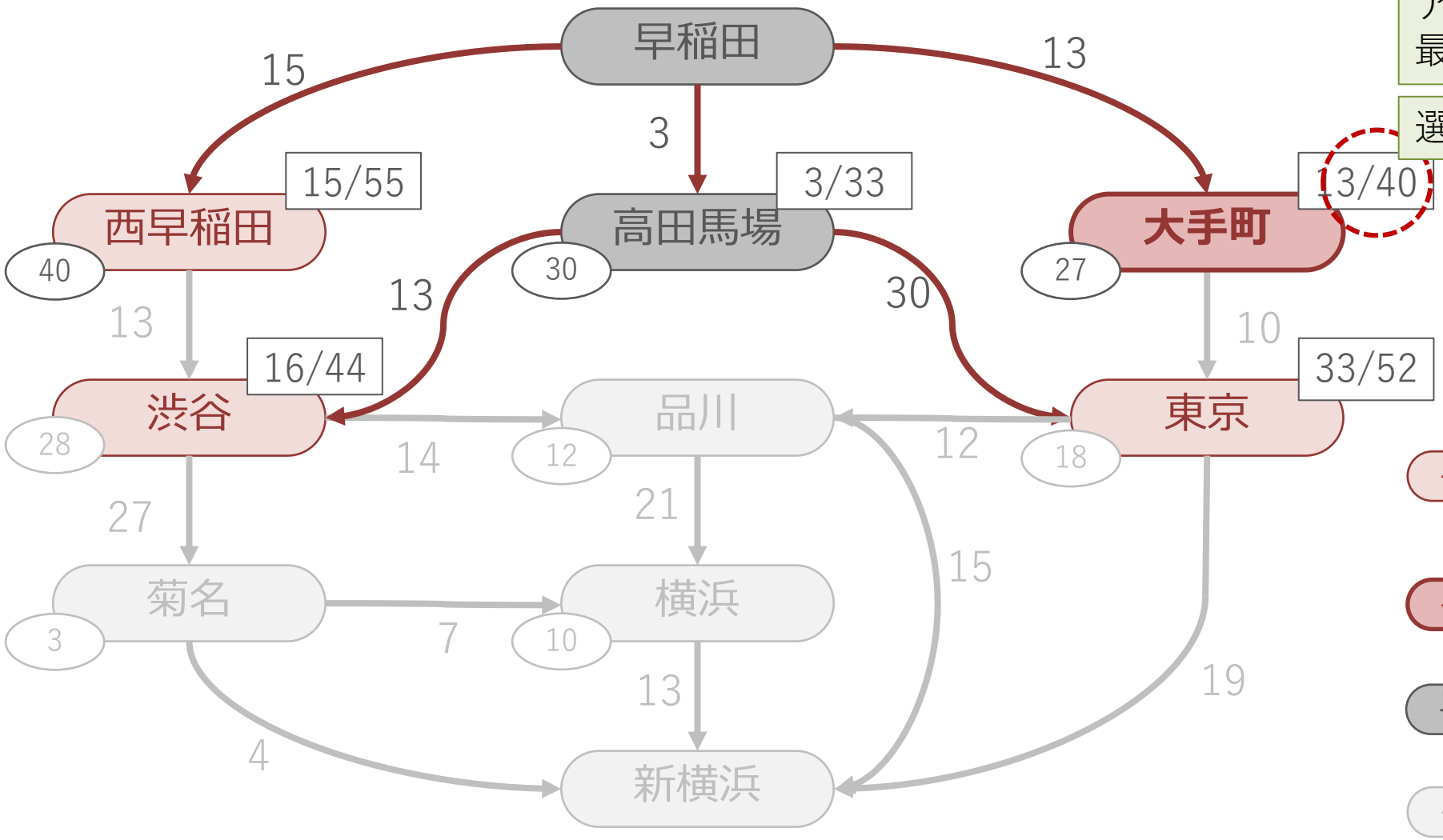
早稲田 アクティブリスト中のノード

早稲田 展開のために選ばれたノード

早稲田 展開済みのノード

早稲田 アクティブになっていないノード

h(n) (ゴールに至るコストの予想値）を用いる場合



アクティブリスト中の
最良評価値を持つノードを選ぶ

選んだノードを展開する

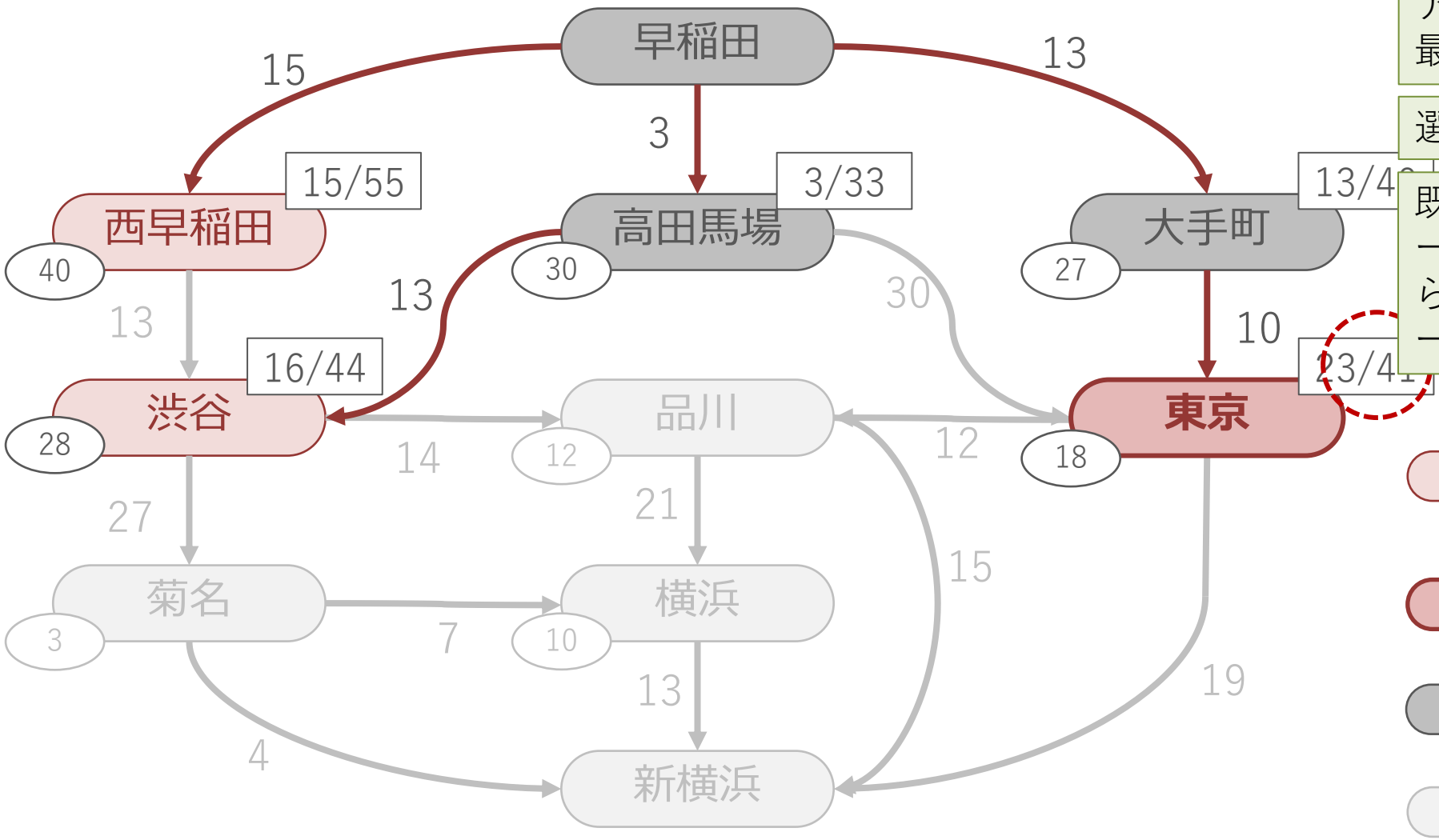
早稲田 アクティブリスト中のノード

早稲田 展開のために選ばれたノード

早稲田 展開済みのノード

早稲田 アクティブになっていないノード

h(n) (ゴールに至るコストの予想値）を用いる場合



アクティブリスト中の
最良評価値を持つノードを選ぶ

選んだノードを展開する

既にアクティブリストにあるノードは，既発見のルートとどちらが良いかを調べ，良い方のルートを残す

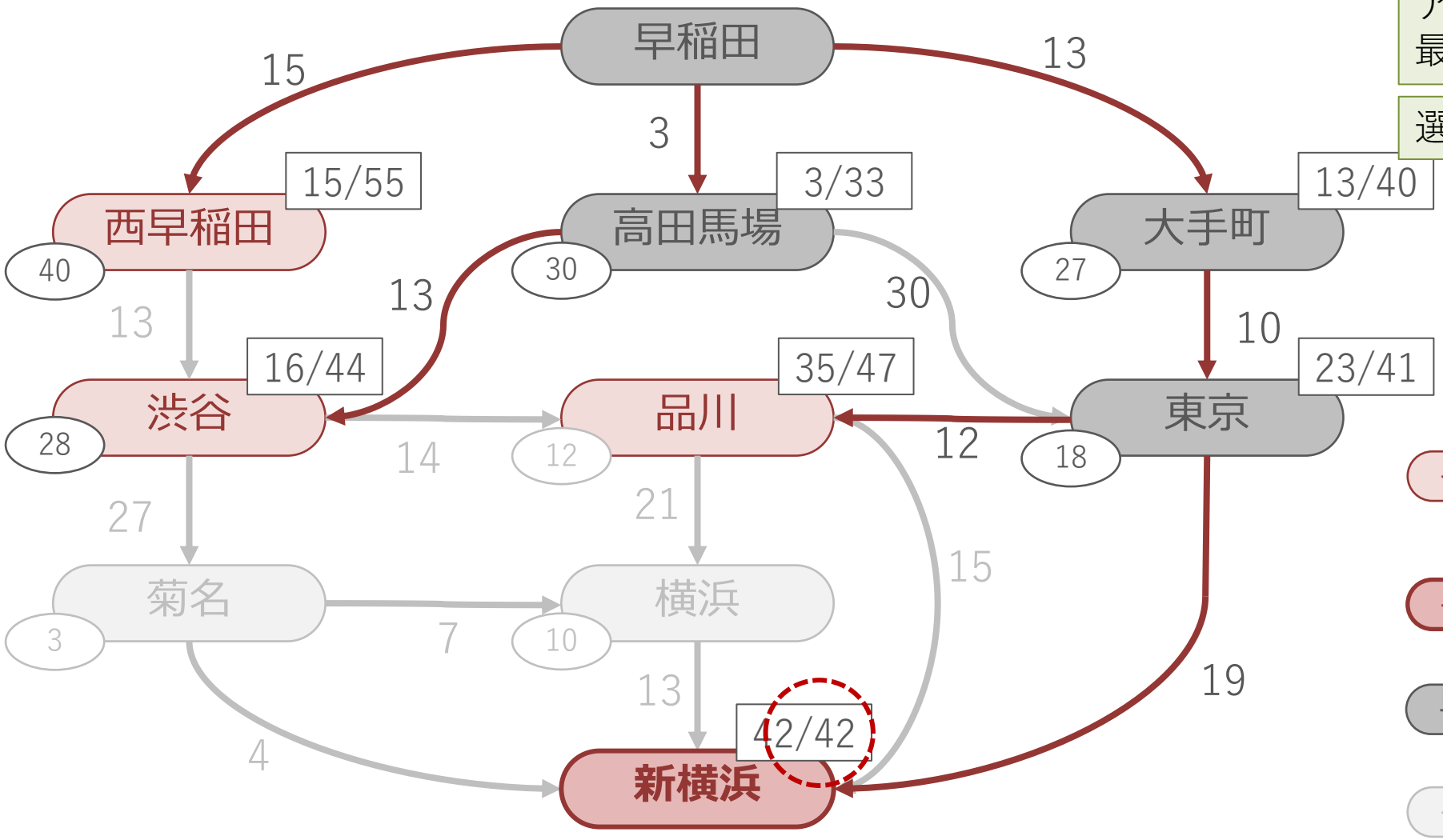
早稲田 アクティブリスト中のノード

早稲田 展開のために選ばれたノード

早稲田 展開済みのノード

早稲田 アクティブになっていないノード

h(n) (ゴールに至るコストの予想値）を用いる場合



アクティブリスト中の
最良評価値を持つノードを選ぶ

選んだノードを展開する

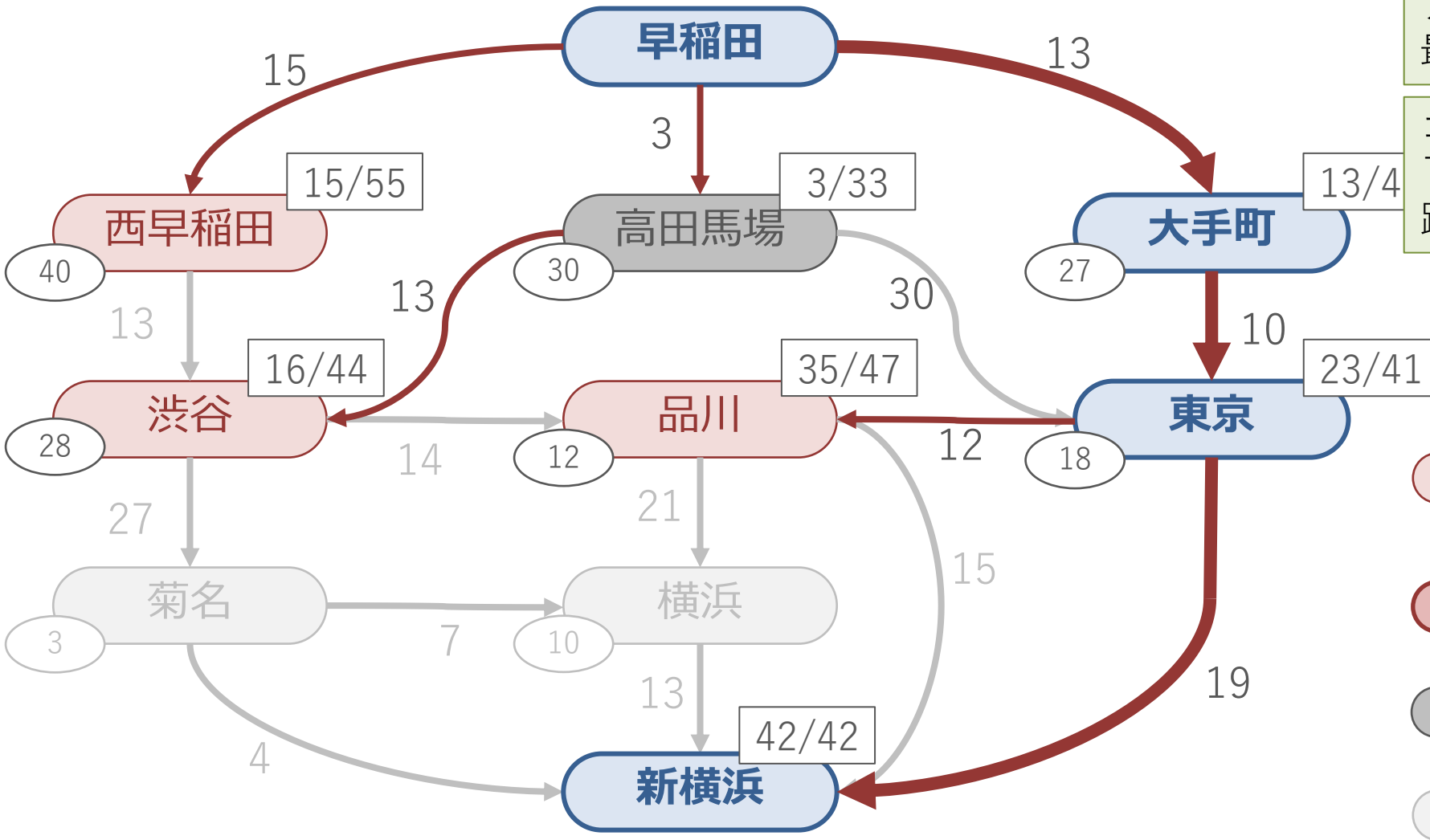
早稲田 アクティブリスト中のノード

早稲田 展開のために選ばれたノード

早稲田 展開済みのノード

早稲田 アクティブになっていないノード

h(n) (ゴールに至るコストの予想値）を用いる場合



アクティブリスト中の
最良評価値を持つノードを選ぶ

ゴールノードが選ばれたら，終了。バックトラックして最適経路を出力。

早稲田 アクティブリスト中のノード

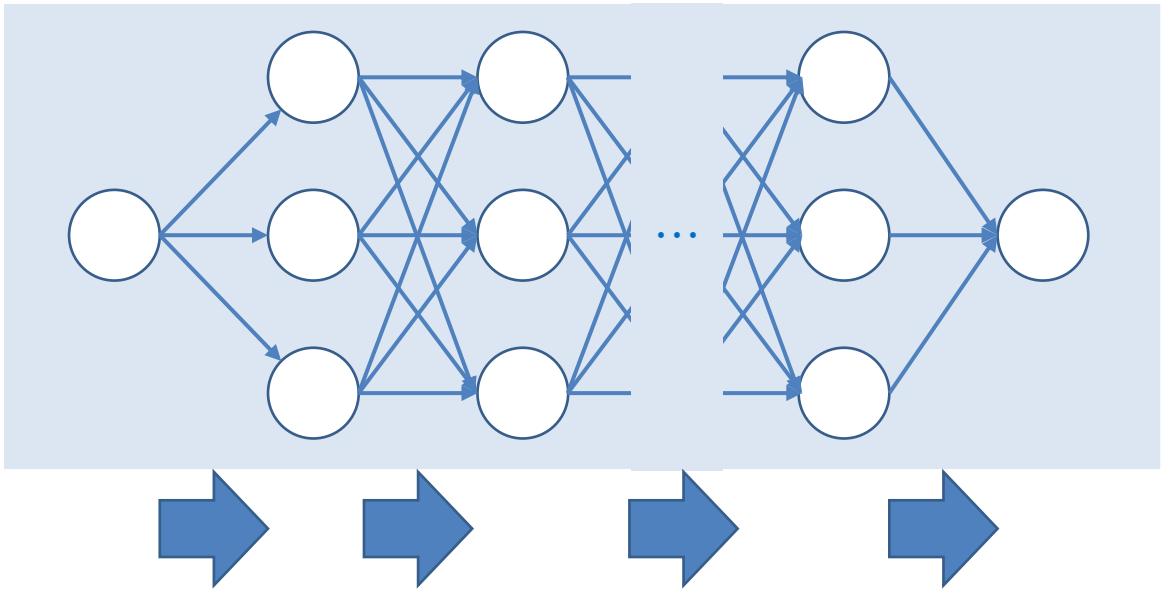
早稲田 展開のために選ばれたノード

早稲田 展開済みのノード

早稲田 アクティブになっていないノード

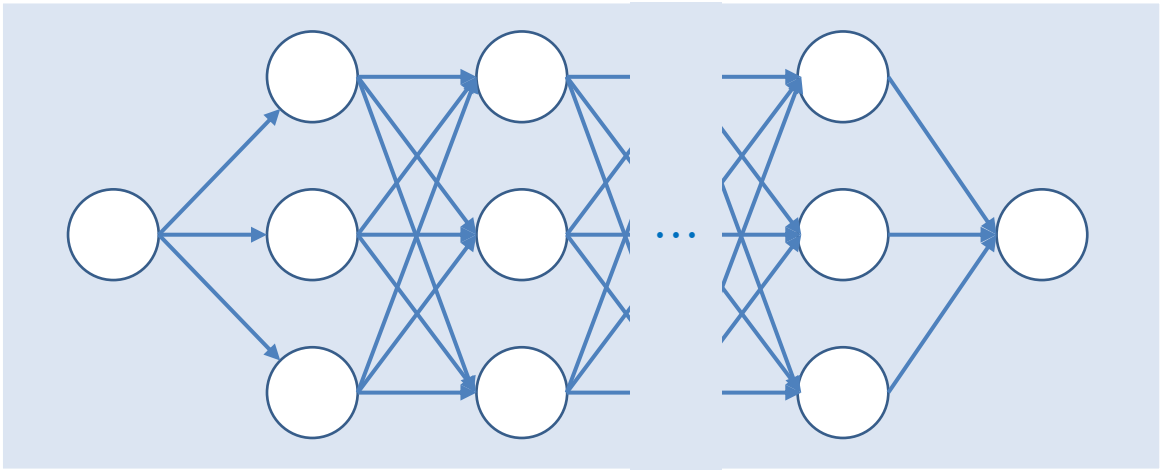
DPとの比較

DP



パスコストが与えられるのに，開始ノードからの経路長に応じた時間遅れがある場合に有効
（状態遷移に時間がかかる場合に有効）
＝次の状態遷移までに可能性のあるものは全て計算しておくのが良い。開始ノードに近いノードのスコア計算を保留するのは時間の無駄。

A＊サーチ



全パスコストが一度に与えられる場合に有効
＝開始ノードに近いノードのスコア計算を保留しておいても時間の無駄にならない。

演習問題

1. 次図において，西早稲田から新横浜まで最短時間で行く経路を，A*サーチで求めることを考える。

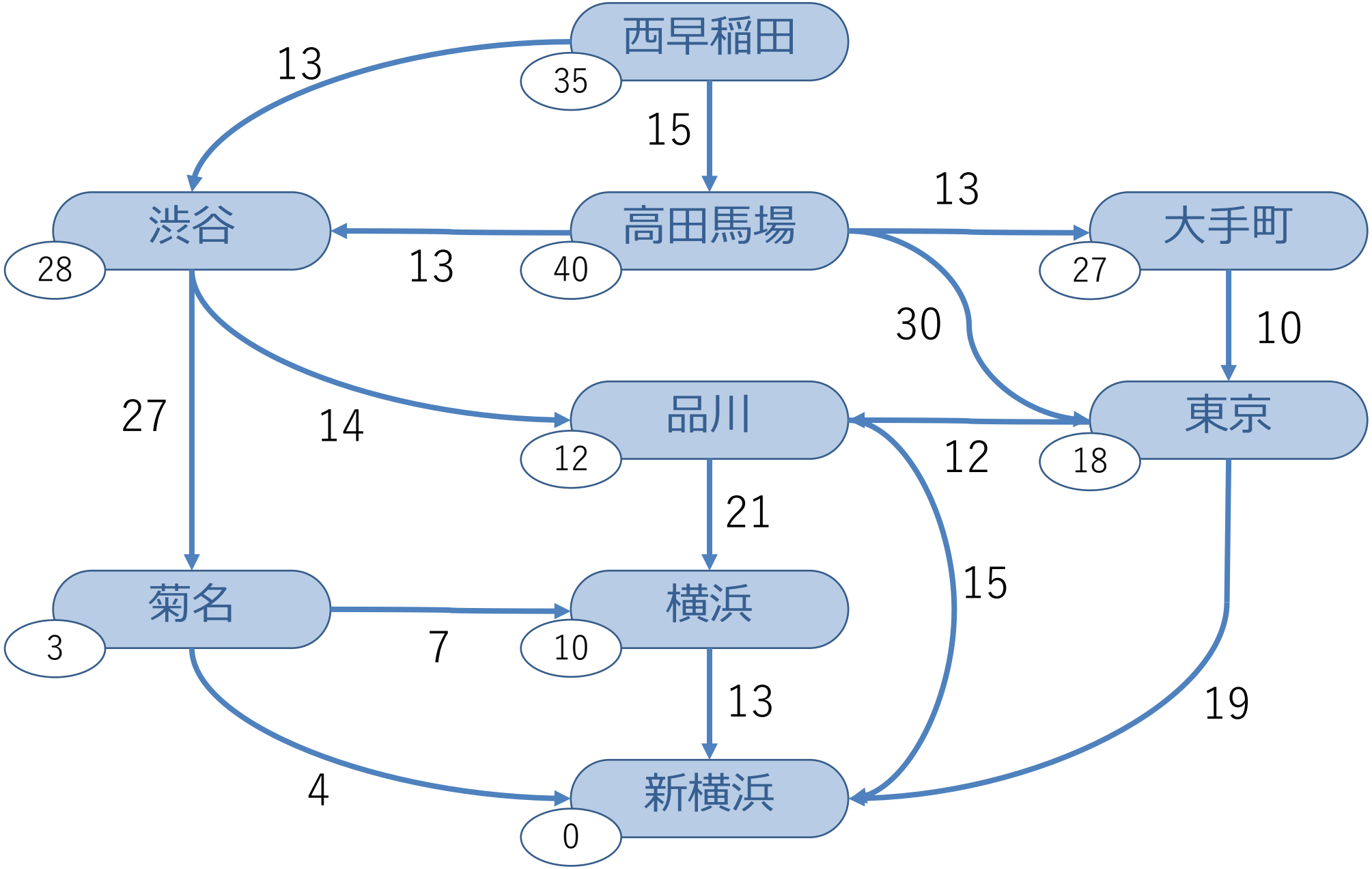
1) 3番目に展開されるノードはどこかを述べよ。

2) 3番目の展開において，アクティブノードに追加されるノードを述べよ。

3) 3番目の展開において，アクティブノードに追加されるノードのうち，最も評価値の良いものと，その評価値を述べよ。

4) 各ノードからゴールまでの推定値（h(n)）を使う場合と，使わない場合のそれぞれについて，展開のために選んだノード（駅）数，アクティブにならなかったノード数を求めよ。

5) 最適解を求めよ。


西早稲田
35
渋谷
28
高田馬場
40
大手町
27
品川
12
東京
18
菊名
3
横浜
10
新横浜
0
13
15
13
13
30
10
27
14
12
21
15
7
13
19
4