応用アルゴリズム演習

Dijkstra’s algorithm/　ダイクストラ法

１月１１日更新→A＊/加点課題説明

7回目　内容

- ダイクストラ法
  - 最短経路問題を紹介
  - 幅優先探索（リマインド）
  - 幅優先探索と比較
  - ダイクストラ法を紹介
- 優先度キュー
  - C言語のqsort・比較手法
  - 実装や詳細→自習
- 課題を紹介

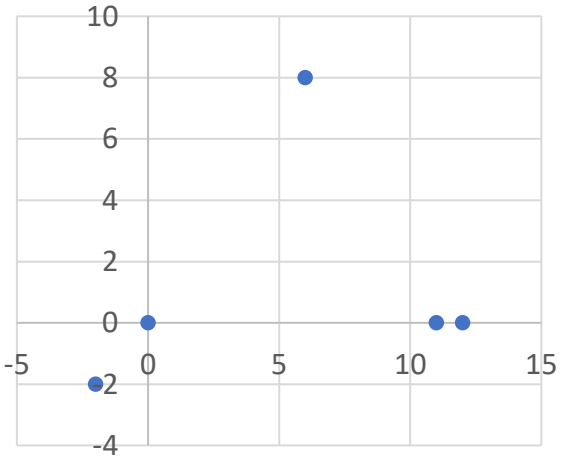
- 次回
  - A＊アルゴリズム
  - 加点課題

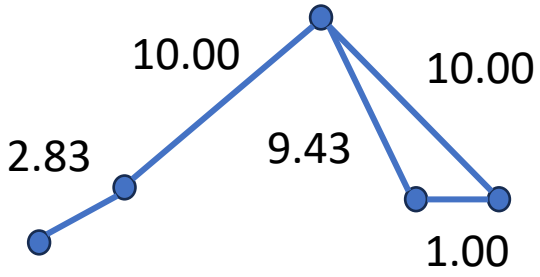
最短経路問題を紹介

盤面に点が配置されています.
始点から終点まで移動したいのですが,
２点間の距離が 10以下の場合だけ移動することができます.
以降で紹介する２種類のアルゴリズムを用いて，最短経路問題を解いてみましょう.

入力例

```
 0  0   スタート
 6  8
-2 -2
11  0
12  0   ゴール
```



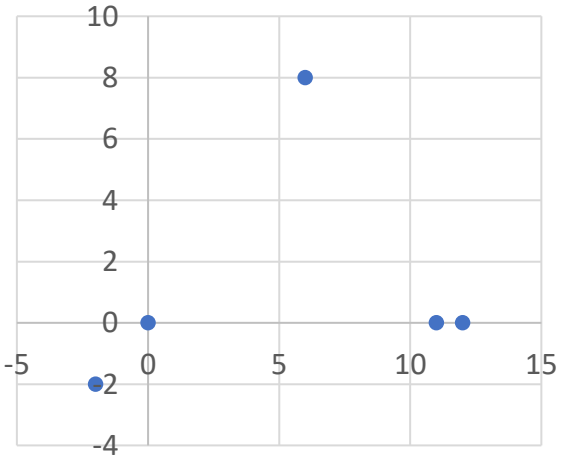


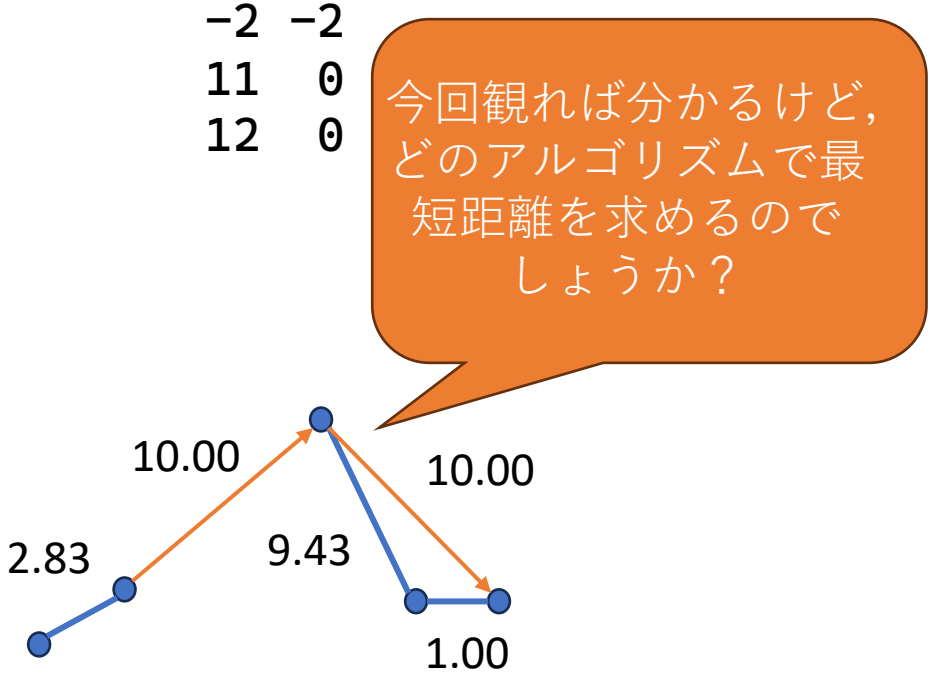
最短経路問題を紹介

盤面に点が配置されています.
始点から終点まで移動したいのですが,
２点間の距離が 10以下の場合だけ移動することができます.
以降で紹介する２種類のアルゴリズムを用いて，最短経路問題を解いてみましょう.

入力例

```
 0  0   スタート
 6  8
-2 -2
11  0
12  0
```

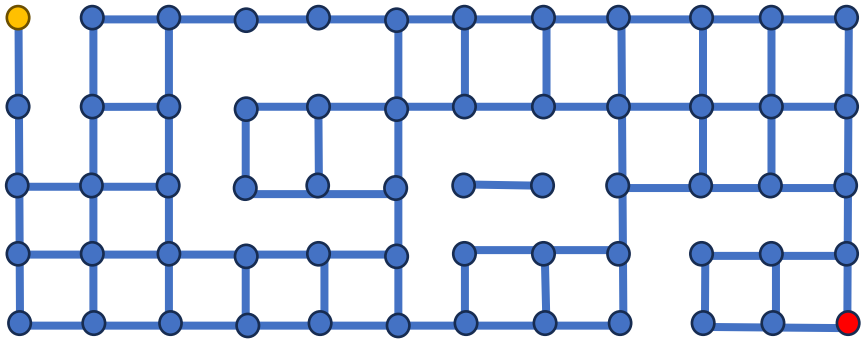




リマインド：幅優先探索

- **よく考えると、グラフの探索ですよね！**

```
int solve( ) {
  enqueue(queue, start);
  while(queue not empty) {
    node_t here = dequeue(queue);
    if(北に行けるか？ && 未訪問) {
      訪問記録
      enqueue(queue, 北へのノード);
    }
    /* 南・西・東同様
  }
  辿りつかなかった場合の処理
  return 答え;
}
```



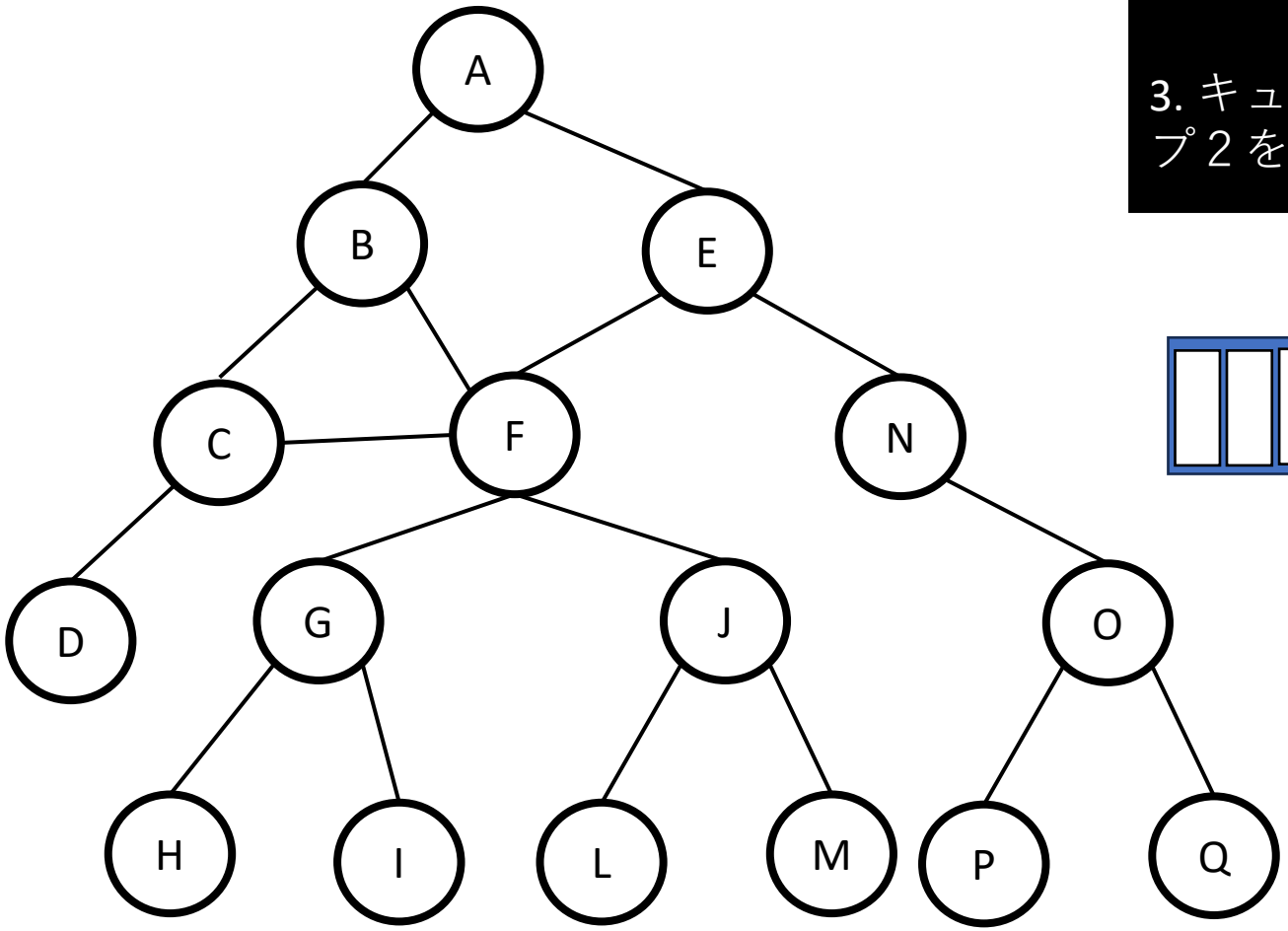
スタート拠点からの距離を記録しながら幅優先探索すると問題解決！

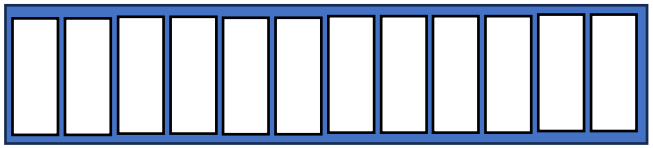
幅優先探索

アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す



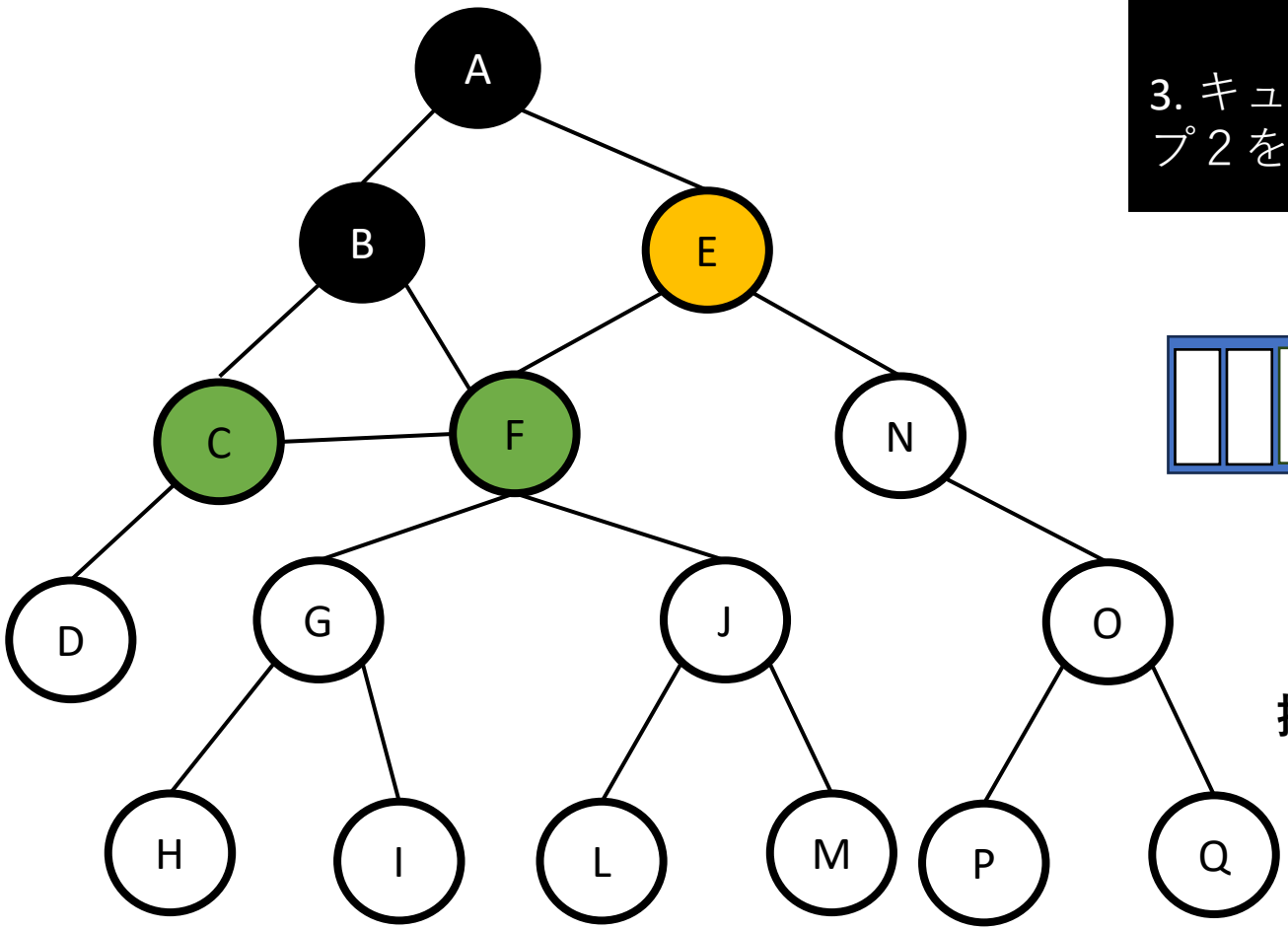


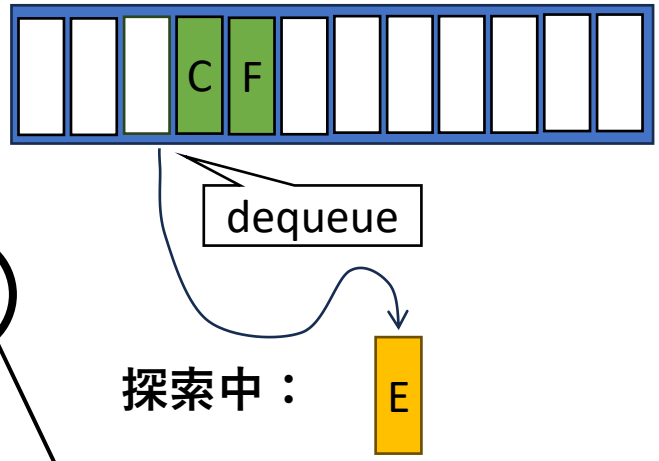
幅優先探索

アルゴリズムの流れ
1. 最初のノードをキューに入れる

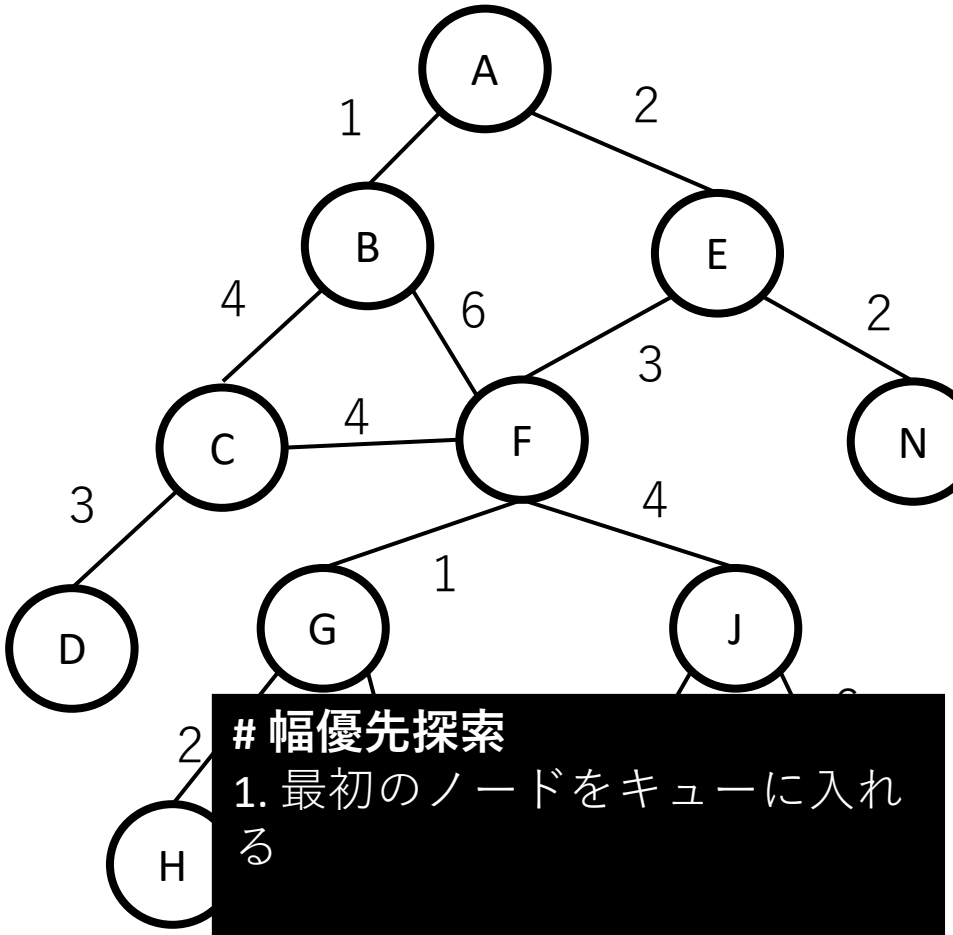
2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す





ダイクストラ法



幅優先探索と比較

- 枝は重み付け

発見したノードへさらに短いルートを見つかる可能性があり！

例：A->B->F 先に見つけて
A->E->F は更に短い

 # 幅優先探索

1. 最初のノードをキューに入れる

2. **キュー**からノードを**取り出して**、隣接するかつ**未探索**のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す

幅優先探索をベースにダイクストラ法を実装

- **待ち行列→優先度キュー**
- **ノードをキューに加える条件を調整**
- **キューから要素を取り出した後に要確認**

優先度キューとは？

例 shortestPath/priorityQtest.c

- 要素を定義

```
typedef struct myitem {
  double priority; /* 優先度、低いほうが優先 */
  int id; /* 要素識別用の番号 */
} myitem_t;
```

- 要素の順位を決める関数

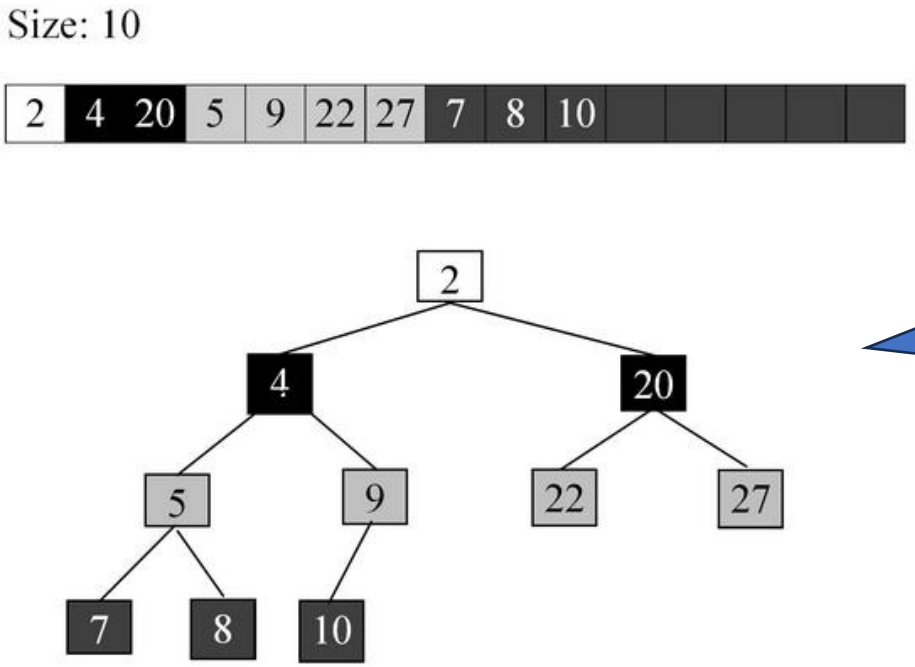
```
/*  * a,b: 要素へのポインタ
 * 返り値： aとb が同じ大きさなら0,
 *          [a, b] の順で並べるべきなら負の値、
 *          [b, a] の順で並べるべきなら正の値を返す
 */
int compare(myitem_t * a, myitem_t * b)
```

Cのスタンダードライブラリが提供しているqsort関数もこれを利用する．詳細はshortestPath/useQsort.c をご覧ください．

優先度キューとは？

例 shortestPath/priorityQtest.c

内部の実装は，１次元配列で木構造



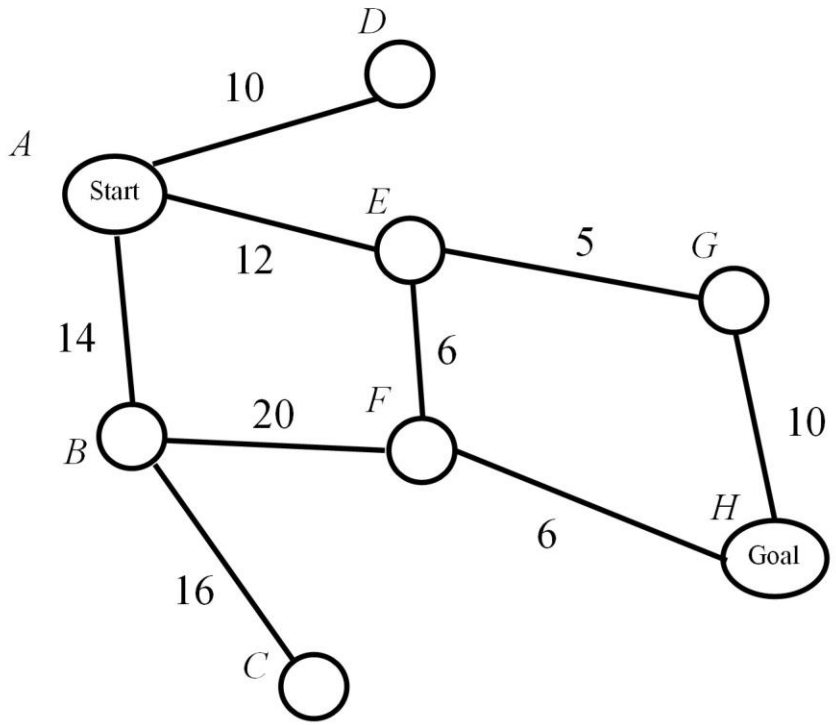
要素を加える・取り出す時に
時間計算量的に効率的！

優先度キュー

- 操作に慣れましょう！

shortestPath/priorityQtest.c を実行して，
要素を**入れた順ではなく，優先度に**取り出していること確認してください

→ダイクストラ法に戻りましょう…

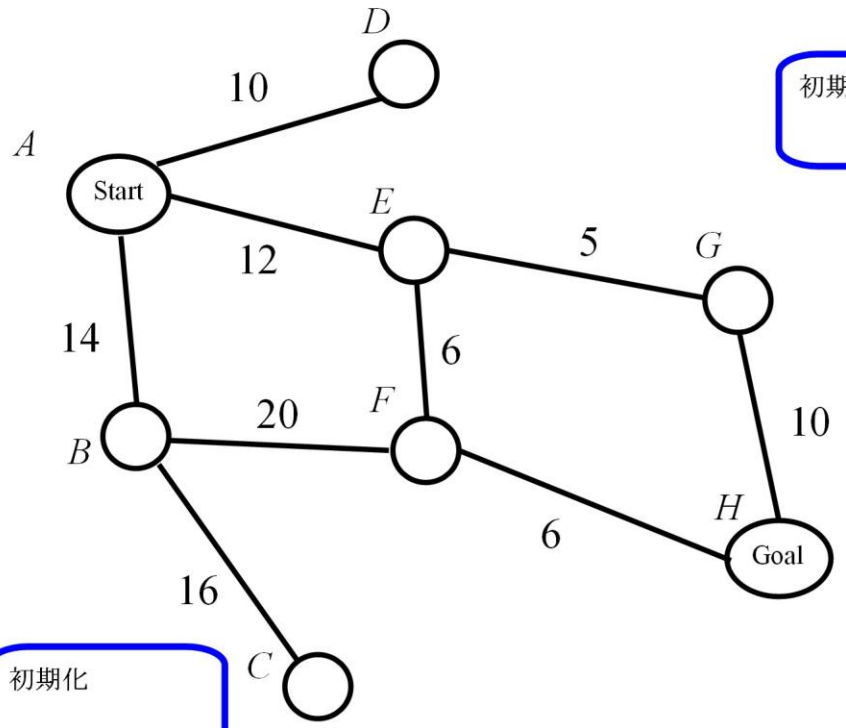


優先度キュー

距離

A	B	C	D	E	F	G	H

探索中ノード



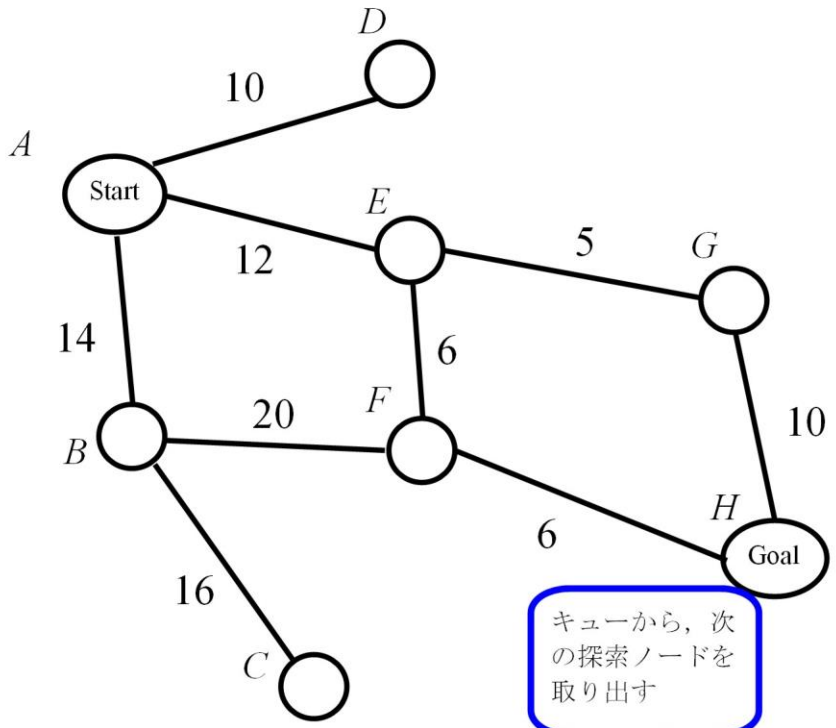
優先度キュー

ノード	優先度
A	0

距離

A	B	C	D	E	F	G	H
0	MAX	MAX	MAX	MAX	MAX	MAX	MAX

探索中ノード



キューから，次の探索ノードを取り出す

優先度キュー

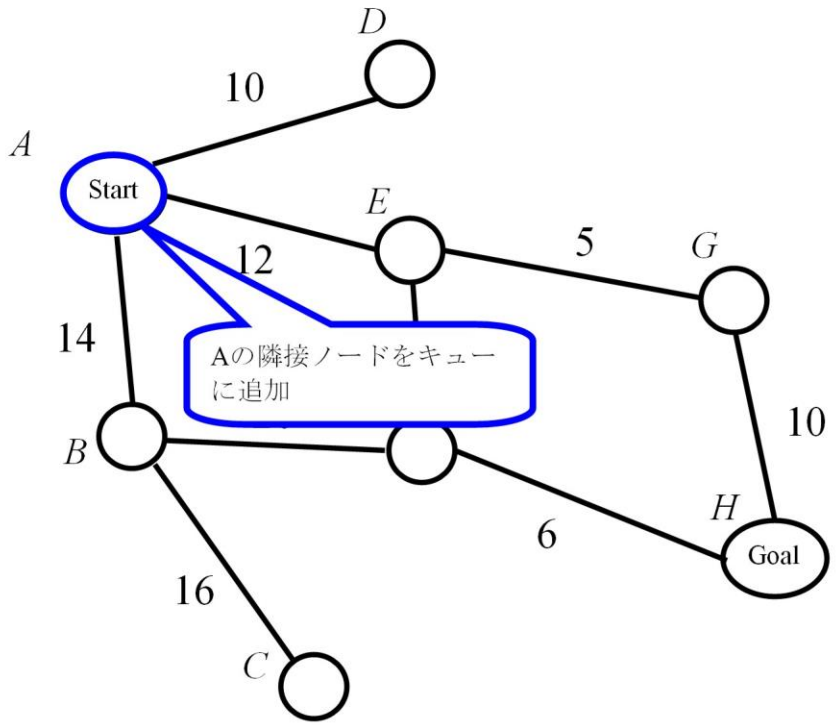
ノード	優先度

探索中ノード

A　0

距離

A	B	C	D	E	F	G	H
0	MAX	MAX	MAX	MAX	MAX	MAX	MAX



優先度キュー

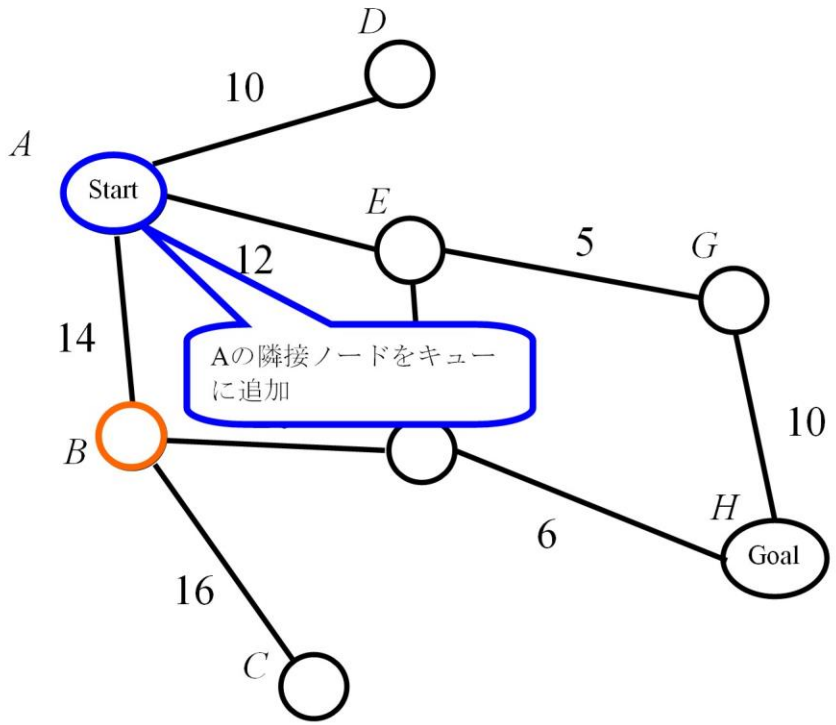
ノード	優先度

距離

A	B	C	D	E	F	G	H
0	MAX	MAX	MAX	MAX	MAX	MAX	MAX

探索中ノード

A　0



優先度キュー

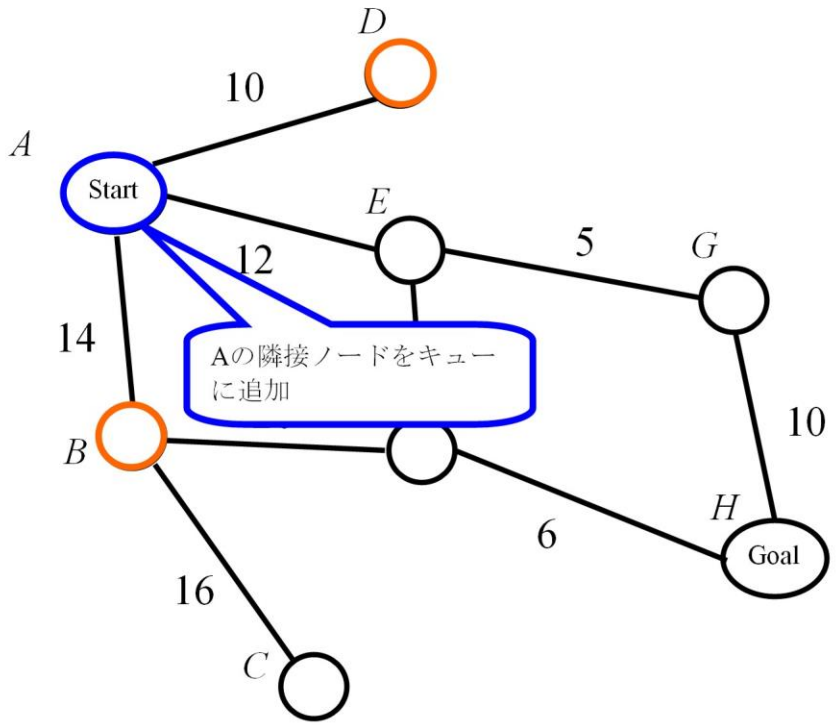
ノード	優先度
B	14

距離

A	B	C	D	E	F	G	H
0	14	MAX	MAX	MAX	MAX	MAX	MAX

探索中ノード

A　0



優先度キュー

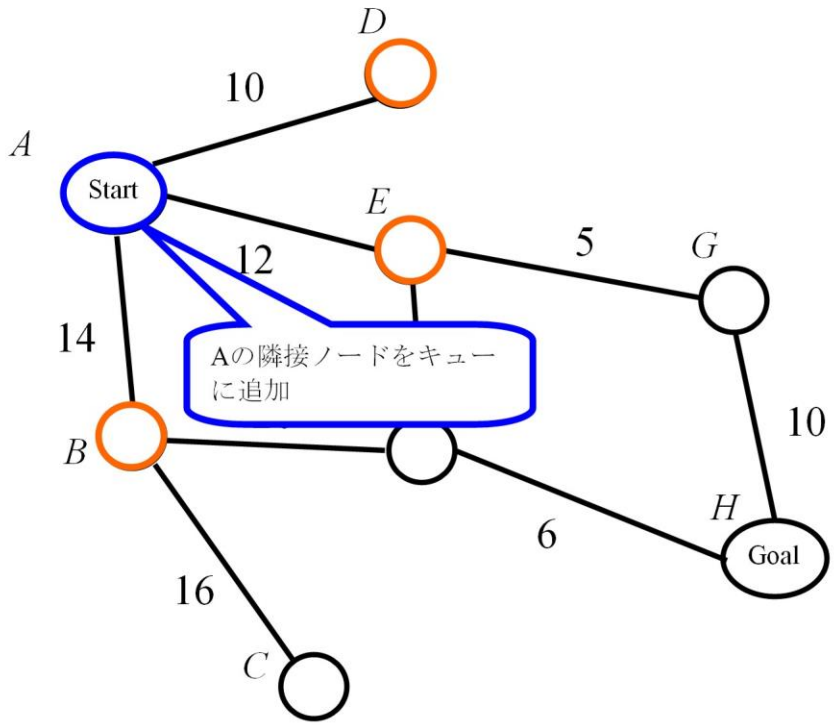
ノード	優先度
D	10
B	14

距離

A	B	C	D	E	F	G	H
0	14	MAX	10	MAX	MAX	MAX	MAX

探索中ノード

A　0



優先度キュー

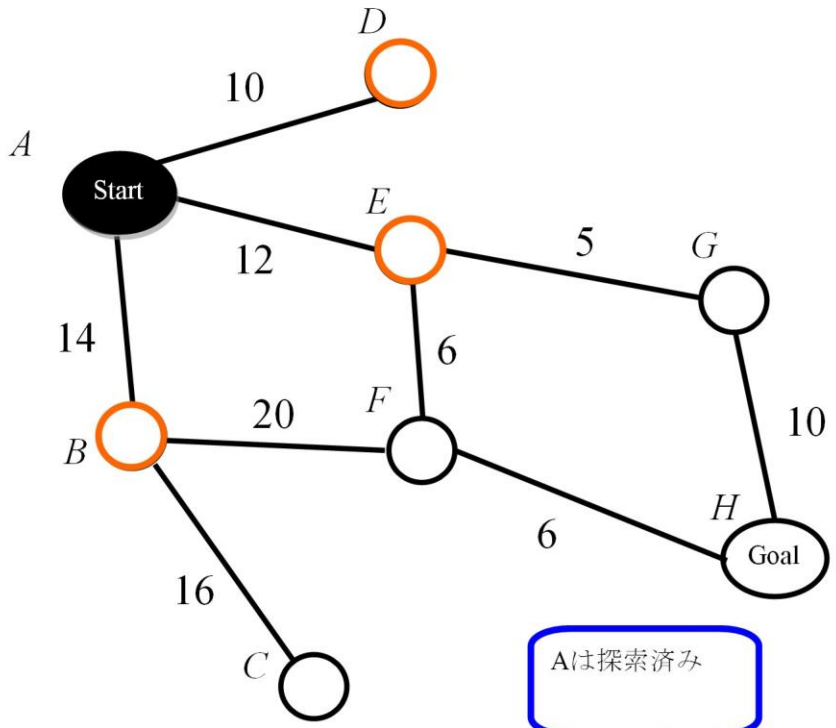
ノード	優先度
D	10
E	12
B	14

距離

A	B	C	D	E	F	G	H
0	14	MAX	10	12	MAX	MAX	MAX

探索中ノード

A　0



Aは探索済み

優先度キュー

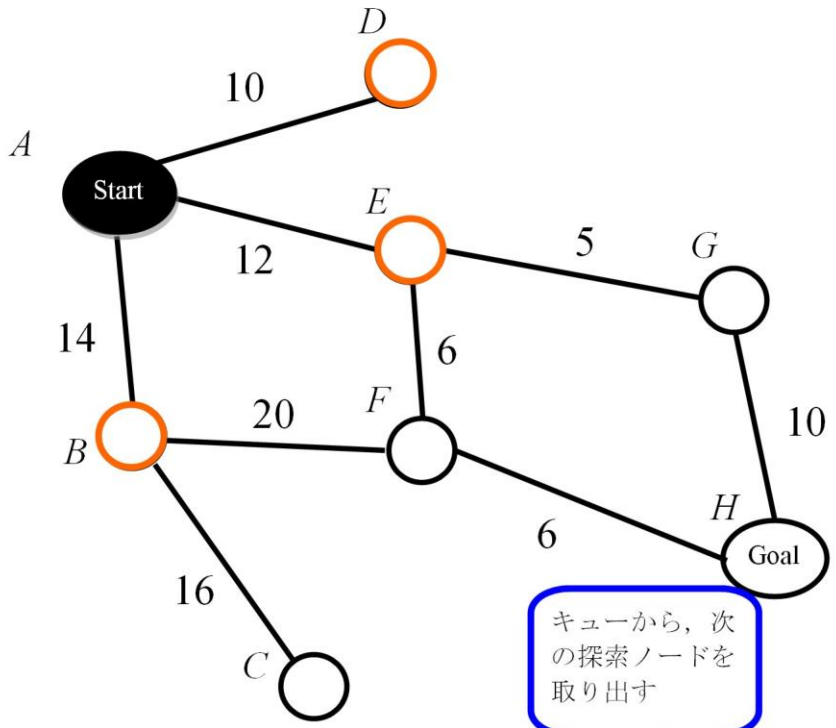
ノード	優先度
D	10
E	12
B	14

距離

A	B	C	D	E	F	G	H
0	14	MAX	10	12	MAX	MAX	MAX

探索中ノード

A　0



キューから，次の探索ノードを取り出す

優先度キュー

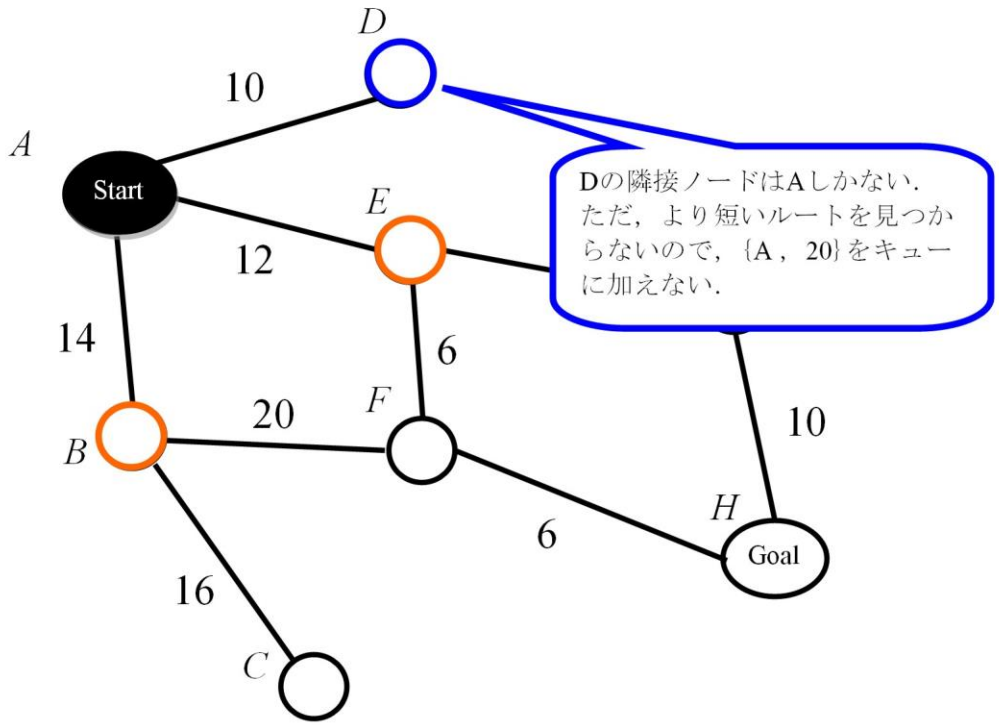
ノード	優先度
E	12
B	14

距離

A	B	C	D	E	F	G	H
0	14	MAX	10	12	MAX	MAX	MAX

探索中ノード

D　10



優先度キュー

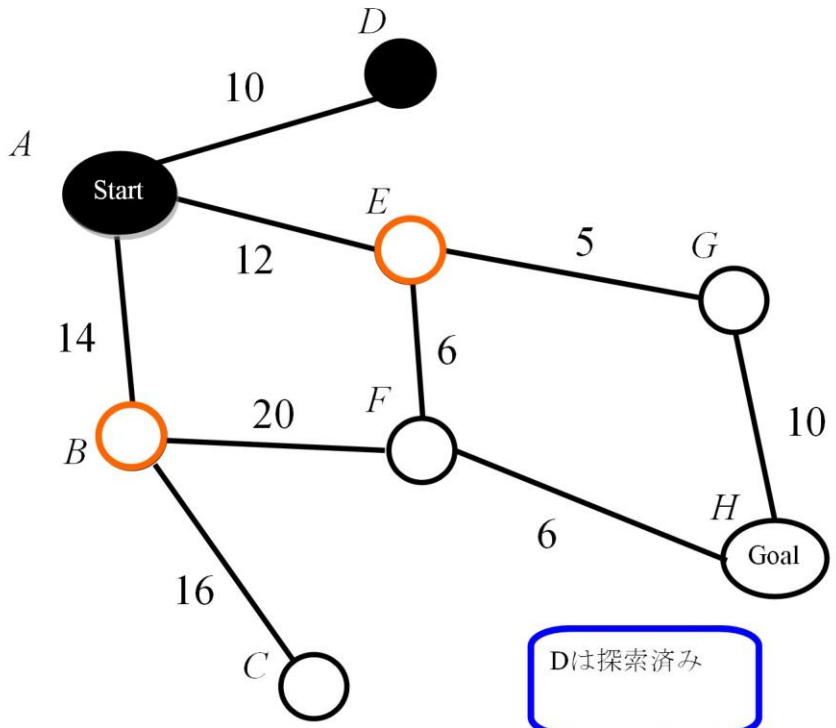
ノード	優先度
E	12
B	14

距離

A	B	C	D	E	F	G	H
0	14	MAX	10	12	MAX	MAX	MAX

探索中ノード

D　10



Dは探索済み

優先度キュー

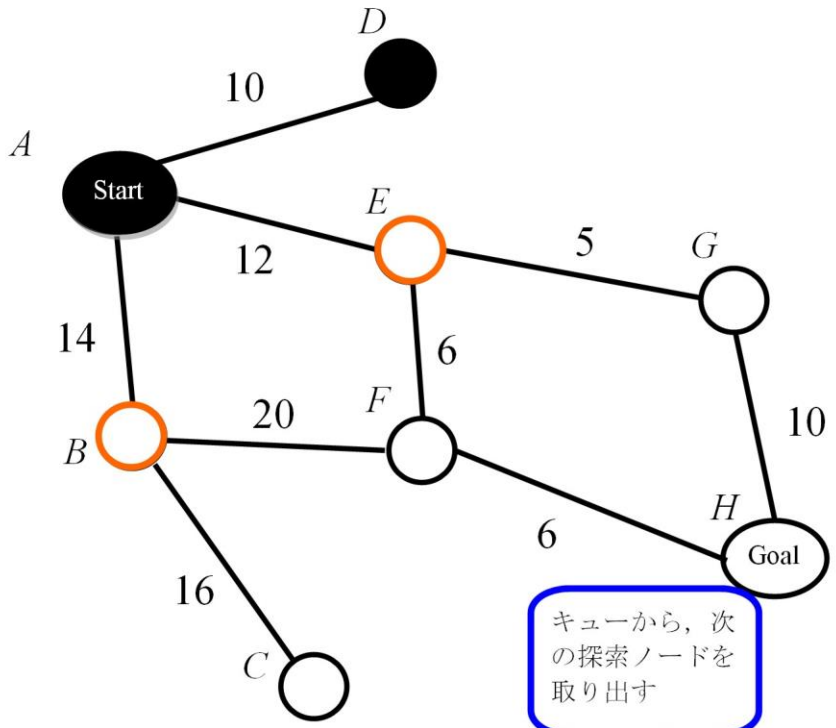
ノード	優先度
E	12
B	14

距離

A	B	C	D	E	F	G	H
0	14	MAX	10	12	MAX	MAX	MAX

探索中ノード

D　10



キューから，次の探索ノードを取り出す

優先度キュー

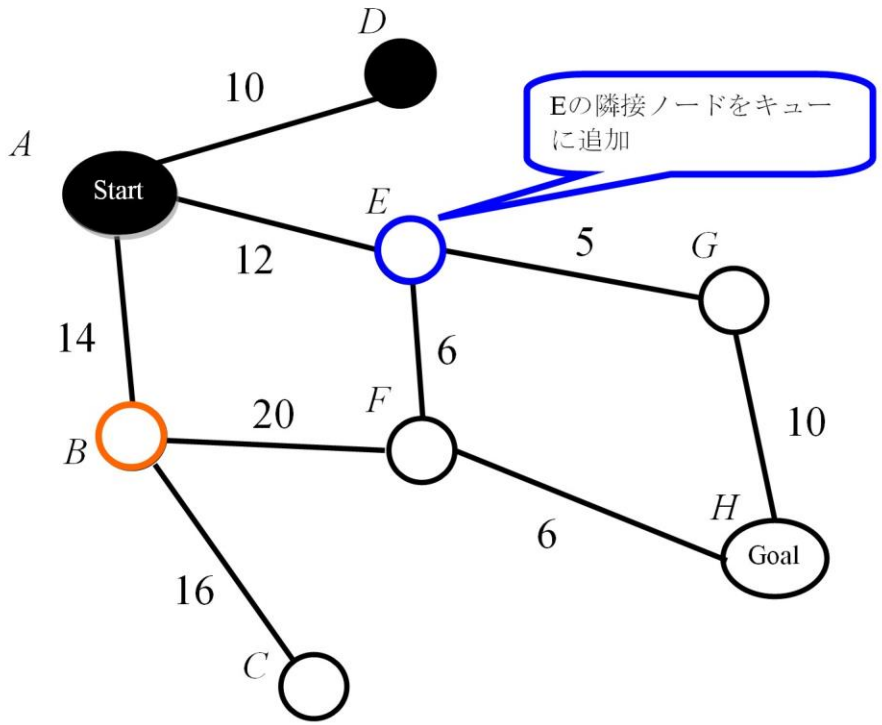
ノード	優先度
B	14

距離

A	B	C	D	E	F	G	H
0	14	MAX	10	12	MAX	MAX	MAX

探索中ノード

E　12



距離

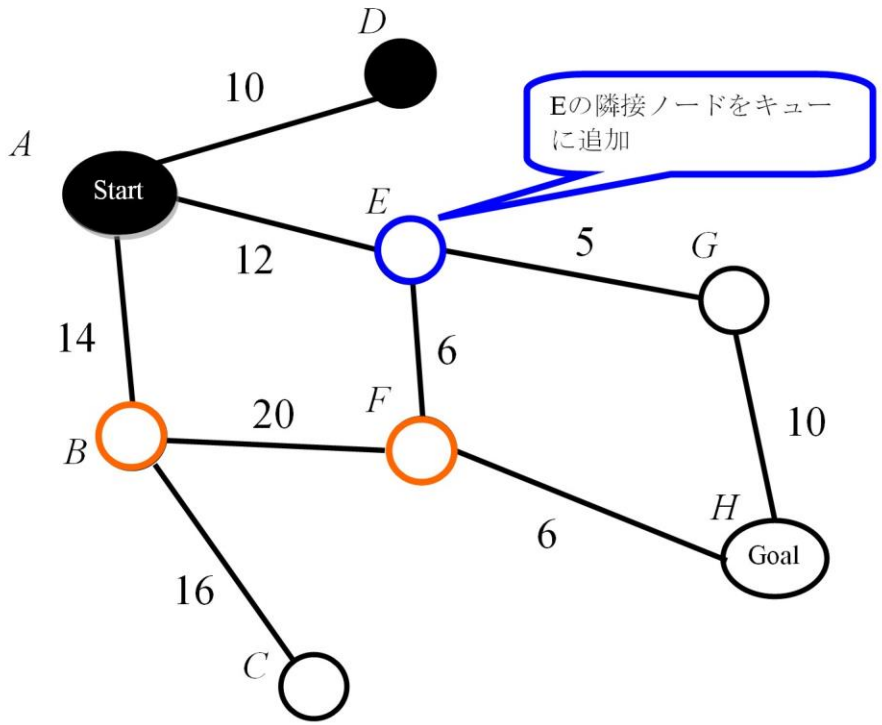
A	B	C	D	E	F	G	H
0	14	MAX	10	12	MAX	MAX	MAX

探索中ノード

E　12

優先度キュー

ノード	優先度
B	14



距離

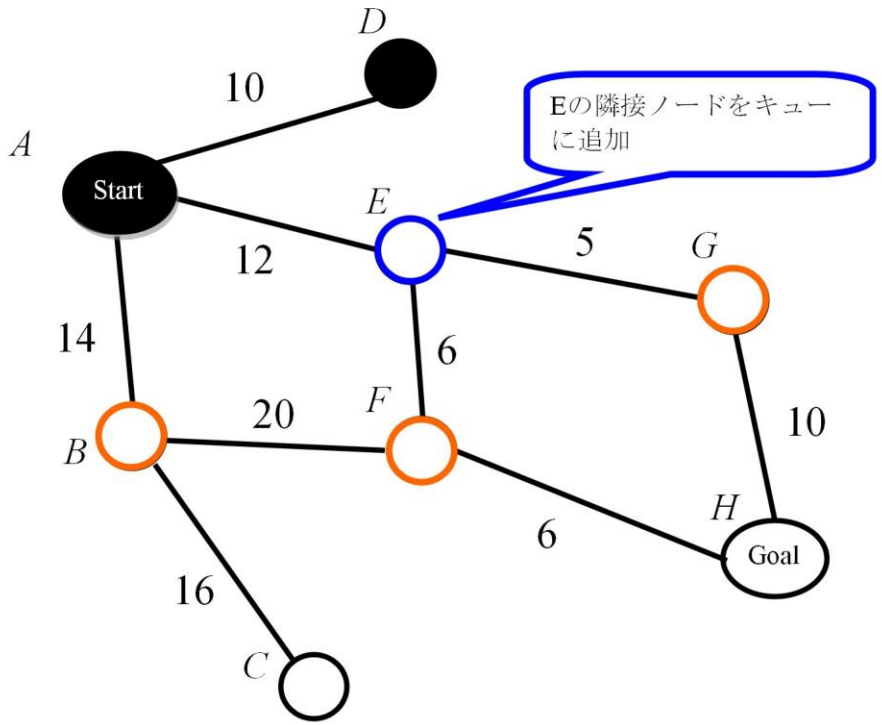
A	B	C	D	E	F	G	H
0	14	MAX	10	12	18	MAX	MAX

探索中ノード

E　12

優先度キュー

ノード	優先度
B	14
F	18



優先度キュー

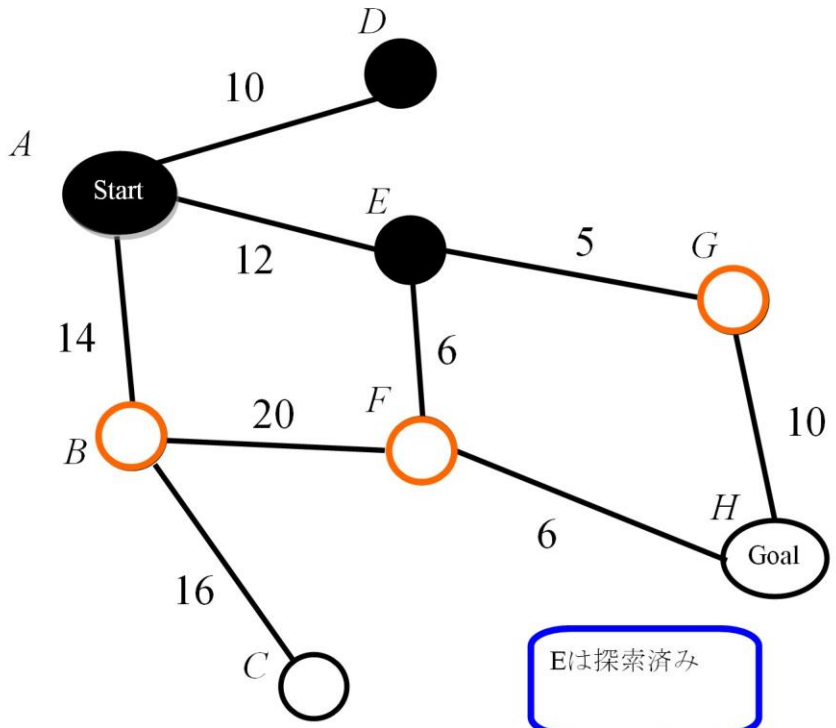
ノード	優先度
B	14
G	17
F	18

距離

A	B	C	D	E	F	G	H
0	14	MAX	10	12	18	17	MAX

探索中ノード

E　12



優先度キュー

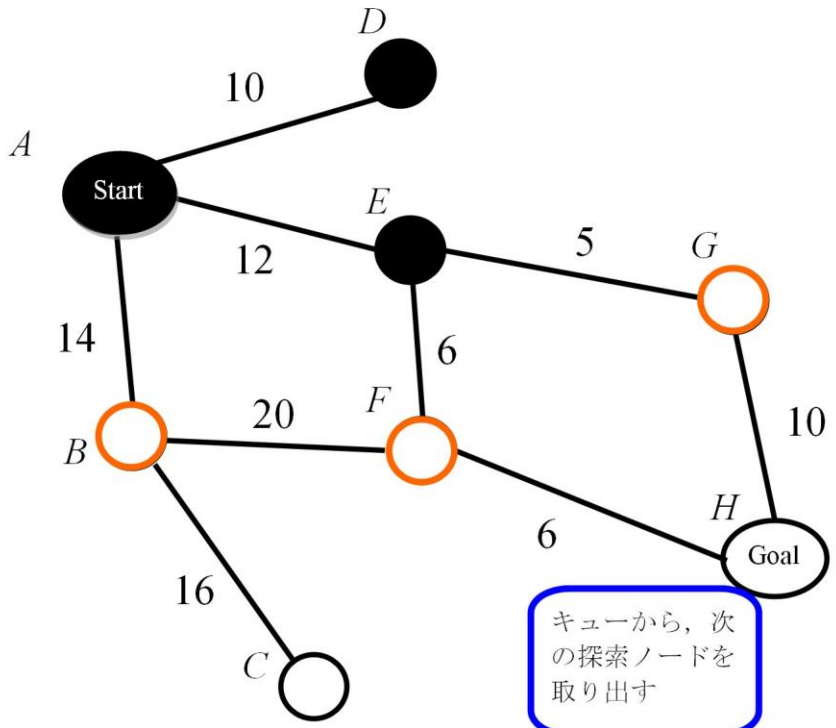
ノード	優先度
B	14
G	17
F	18

距離

A	B	C	D	E	F	G	H
0	14	MAX	10	12	18	17	MAX

探索中ノード

E　12



キューから，次の探索ノードを取り出す

優先度キュー

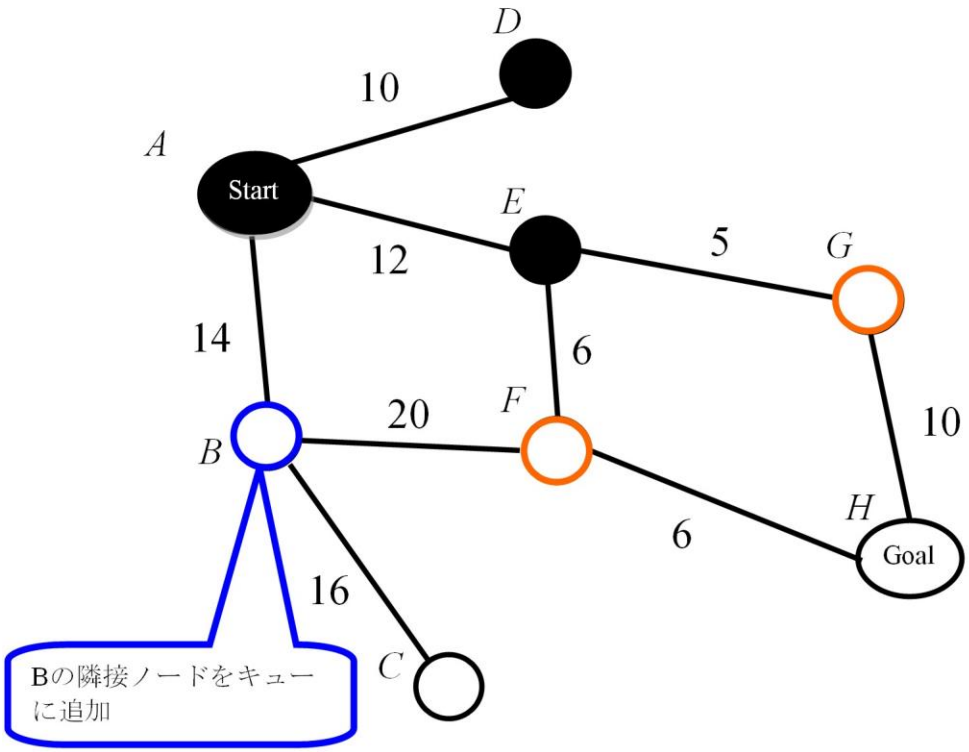
ノード	優先度
G	17
F	18

探索中ノード

B　14

距離

A	B	C	D	E	F	G	H
0	14	MAX	10	12	18	17	MAX



優先度キュー

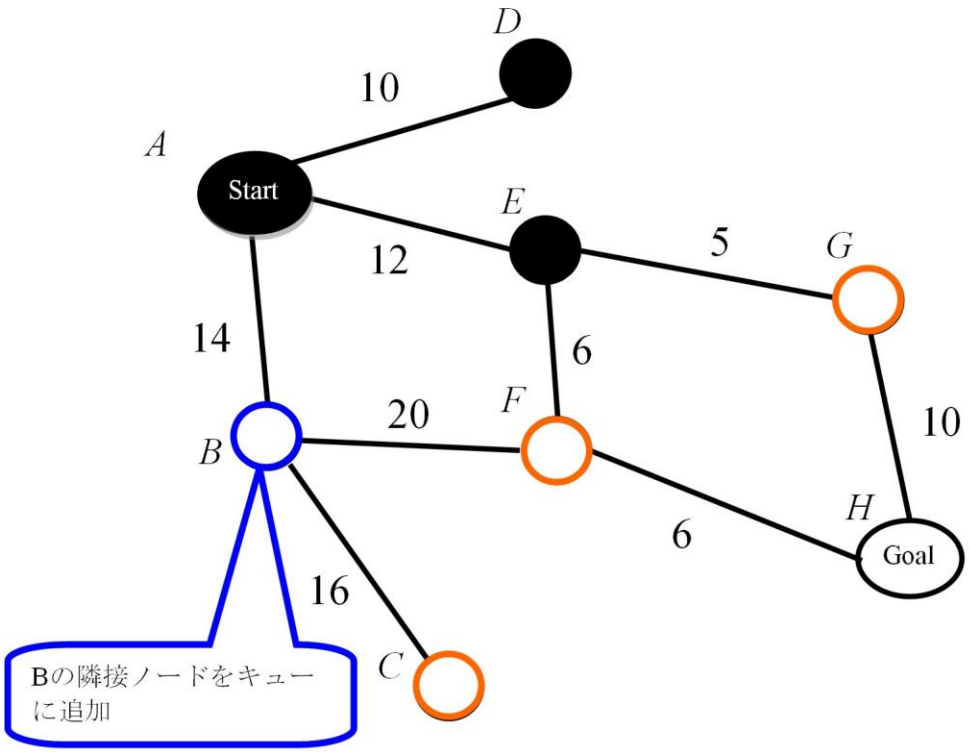
ノード	優先度
G	17
F	18

距離

A	B	C	D	E	F	G	H
0	14	MAX	10	12	18	17	MAX

探索中ノード

B　14



優先度キュー

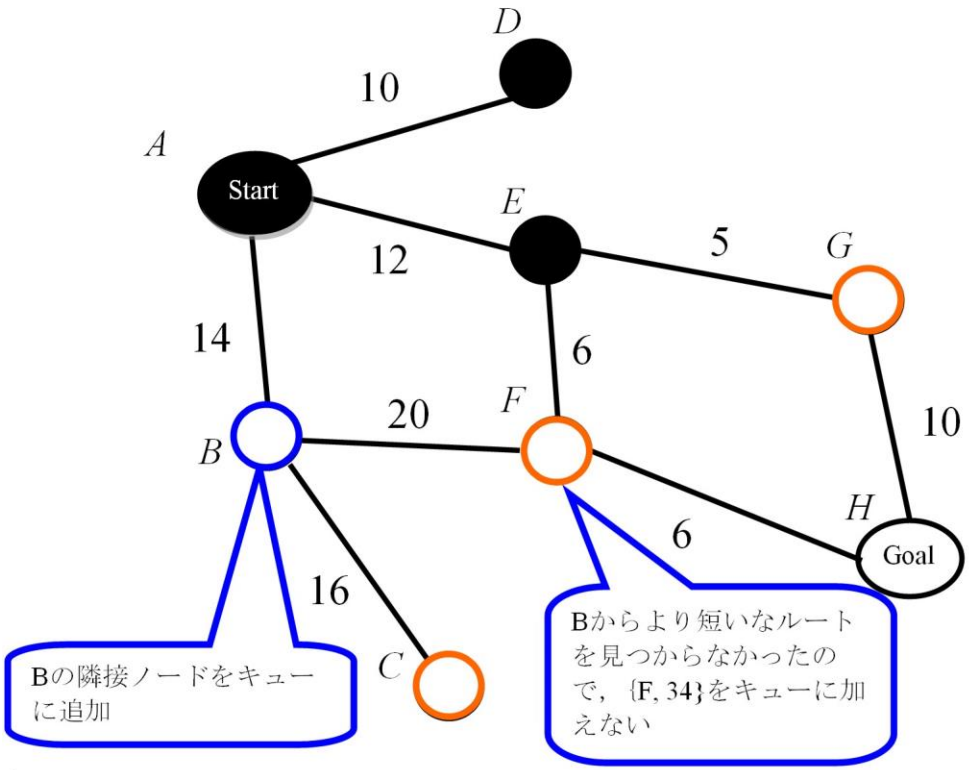
ノード	優先度
G	17
F	18
C	30

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	MAX

探索中ノード

B　14



優先度キュー

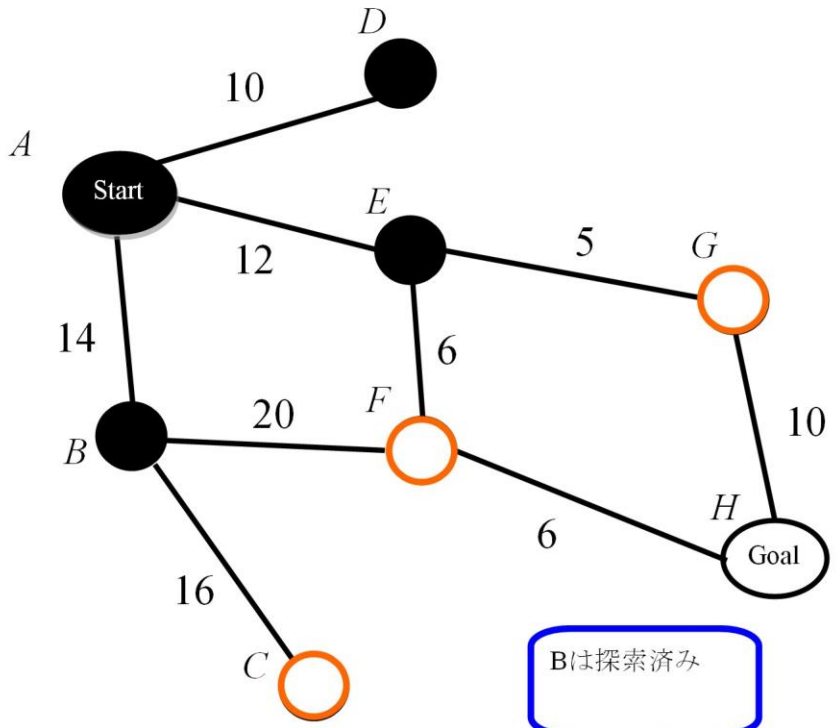
ノード	優先度
G	17
F	18
C	30

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	MAX

探索中ノード

B　14



優先度キュー

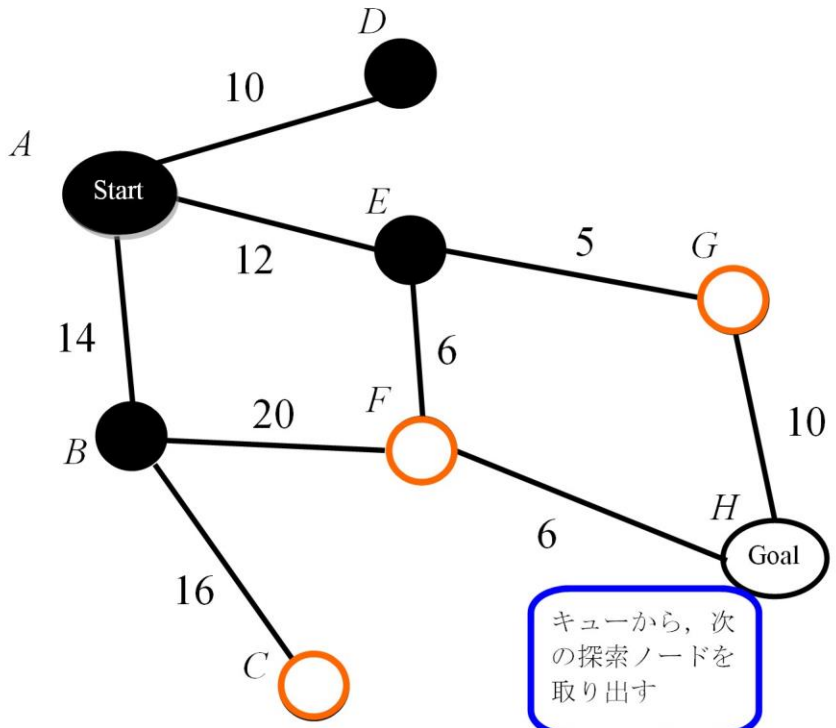
ノード	優先度
G	17
F	18
C	30

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	MAX

探索中ノード

B　14



キューから，次の探索ノードを取り出す

優先度キュー

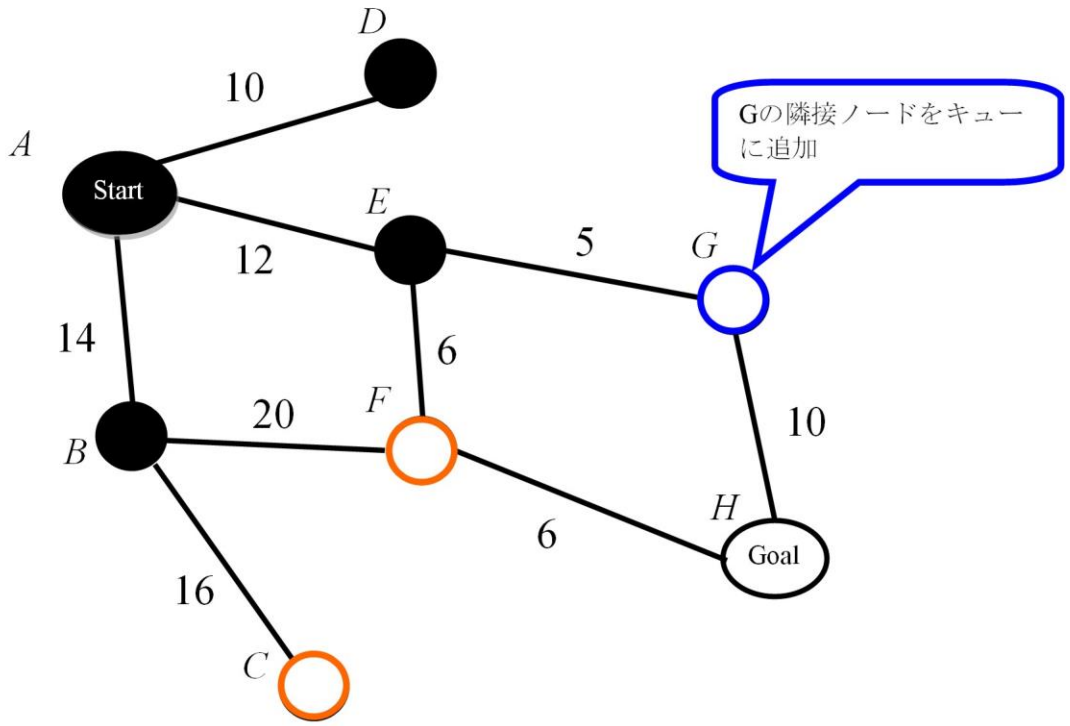
ノード	優先度
F	18
C	30

探索中ノード

G 17

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	MAX



優先度キュー

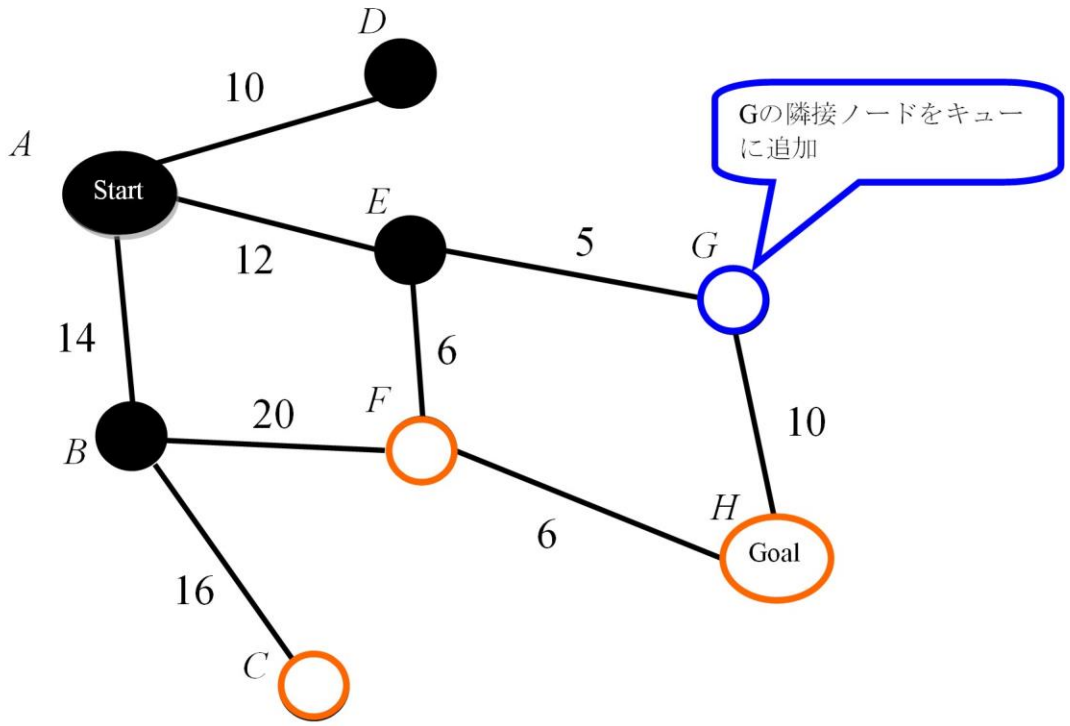
ノード	優先度
F	18
C	30

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	MAX

探索中ノード

G 17



優先度キュー

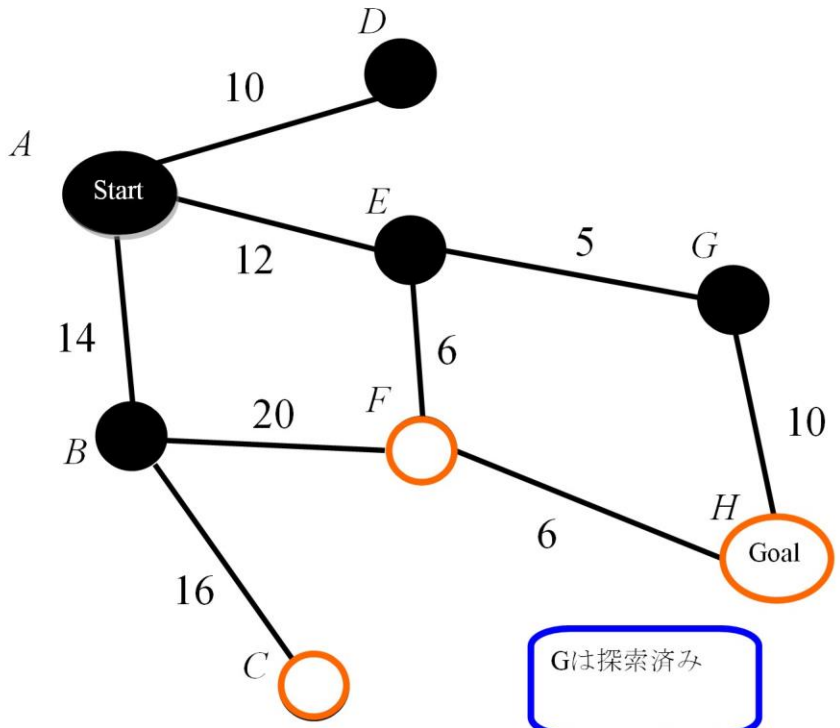
ノード	優先度
F	18
H	27
C	30

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	27

探索中ノード

G 17



優先度キュー

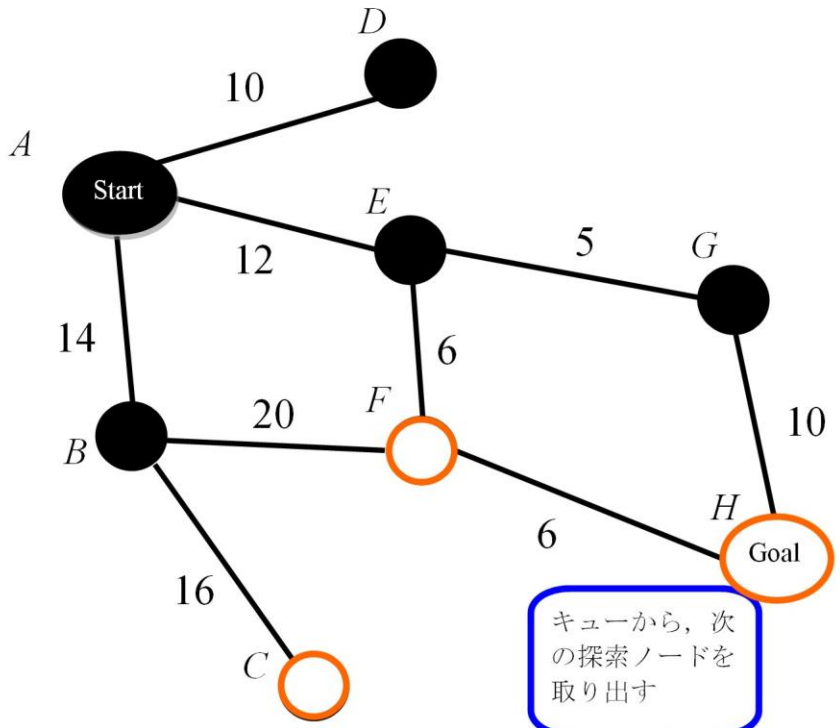
ノード	優先度
F	18
H	27
C	30

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	27

探索中ノード

G 17



優先度キュー

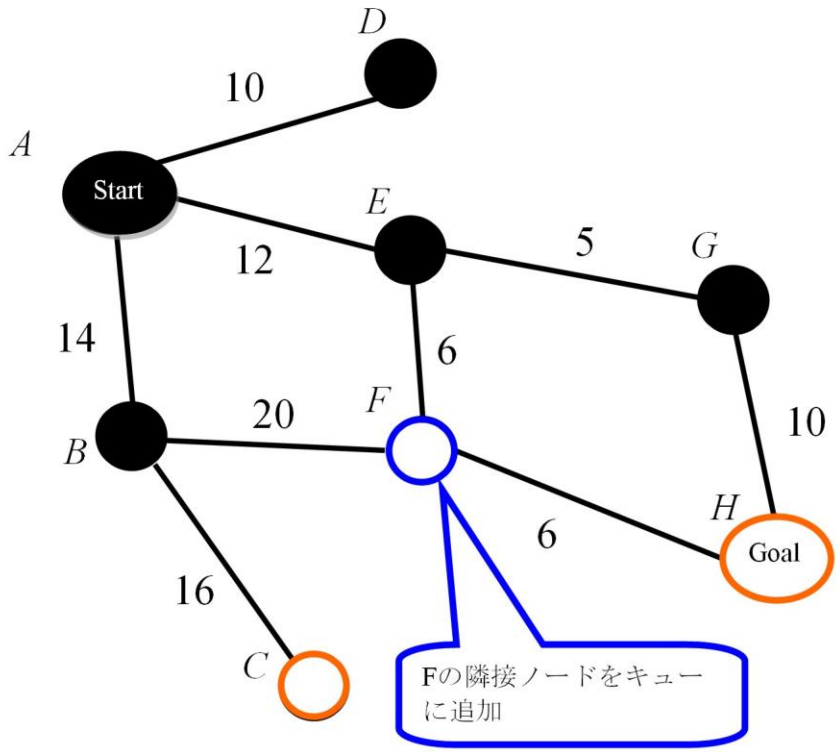
ノード	優先度
H	27
C	30

探索中ノード

F 18

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	27



優先度キュー

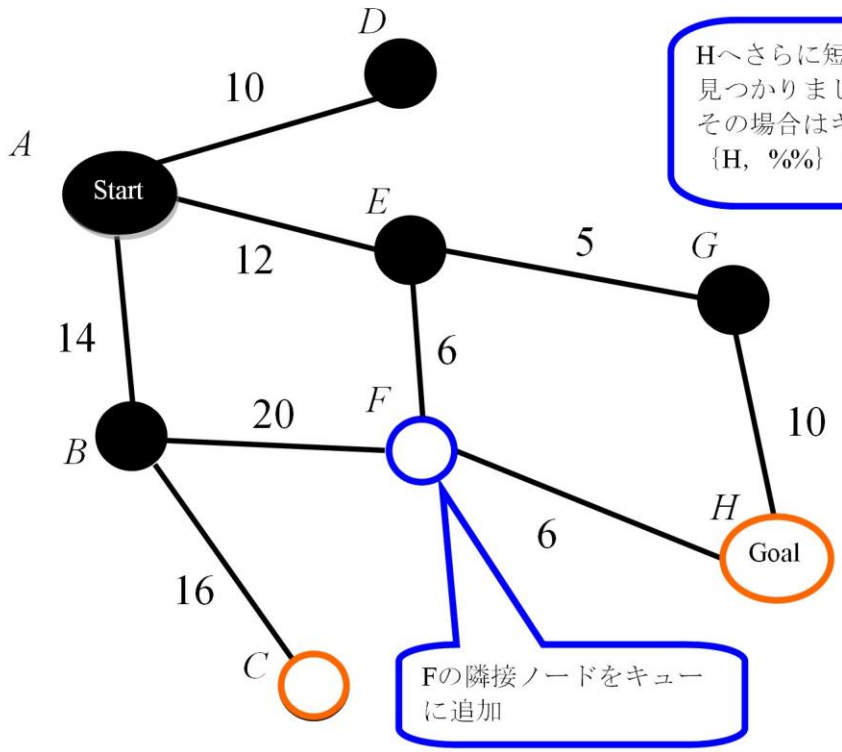
ノード	優先度
H	27
C	30

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	27

探索中ノード

F 18



優先度キュー

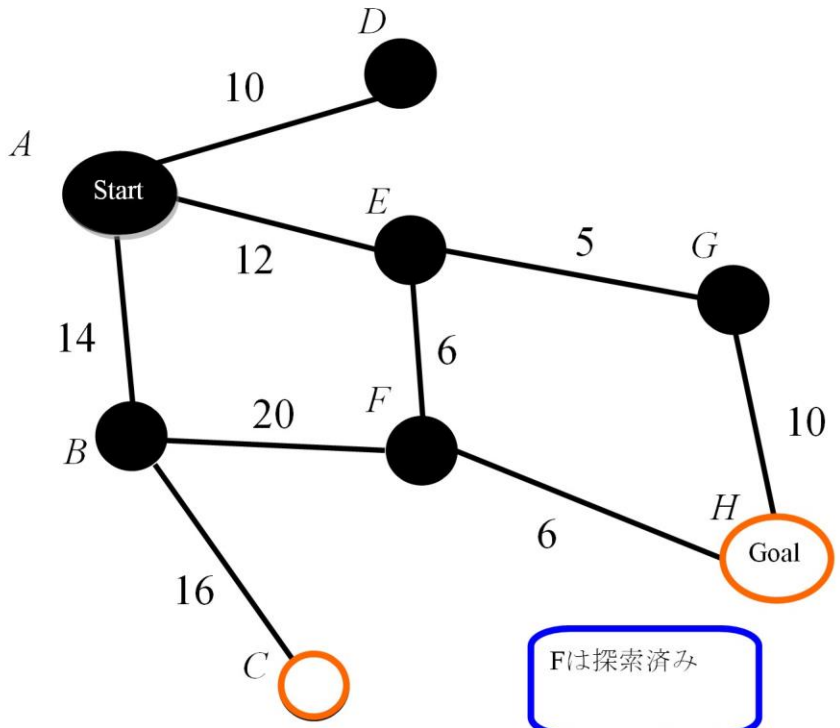
ノード	優先度
H	24
H	27
C	30

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	24

探索中ノード

F 18



優先度キュー

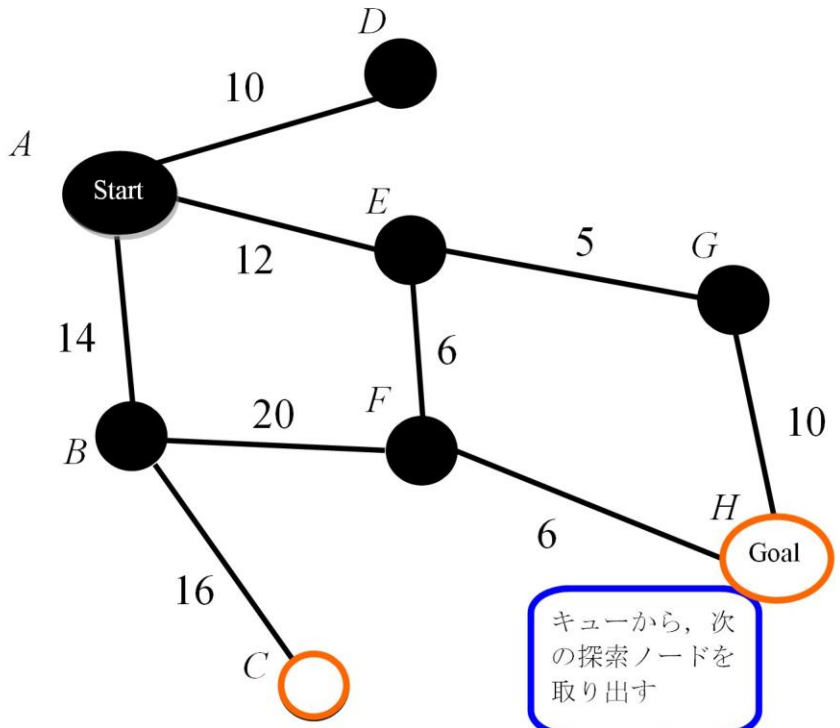
ノード	優先度
H	24
H	27
C	30

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	24

探索中ノード

F 18



キューから，次の探索ノードを取り出す

優先度キュー

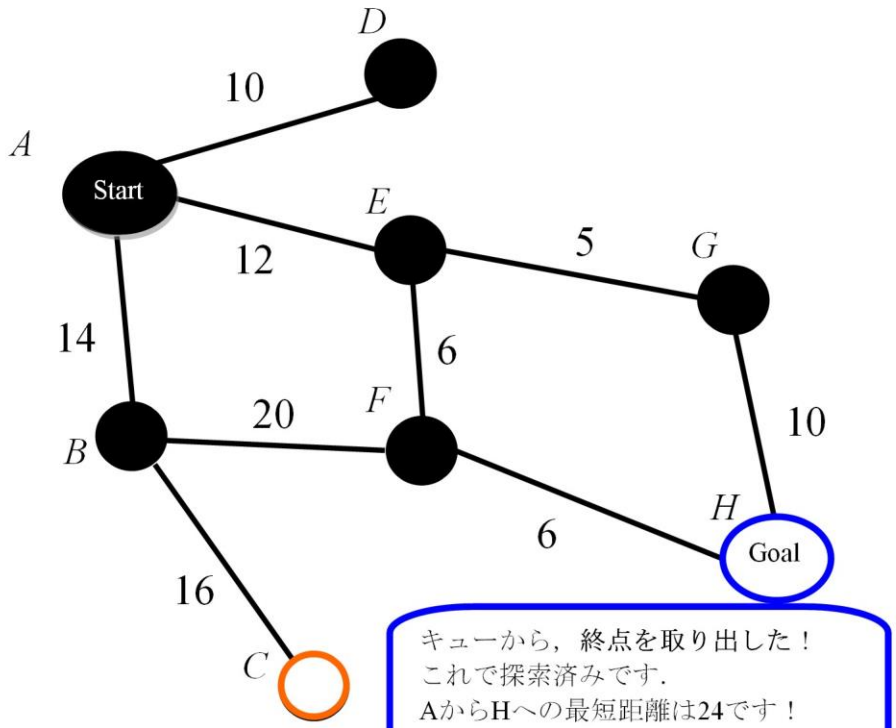
ノード	優先度
H	27
C	30

探索中ノード

H 24

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	24



優先度キュー

ノード	優先度
H	27
C	30

距離

A	B	C	D	E	F	G	H
0	14	30	10	12	18	17	24

索中ノード

H 24

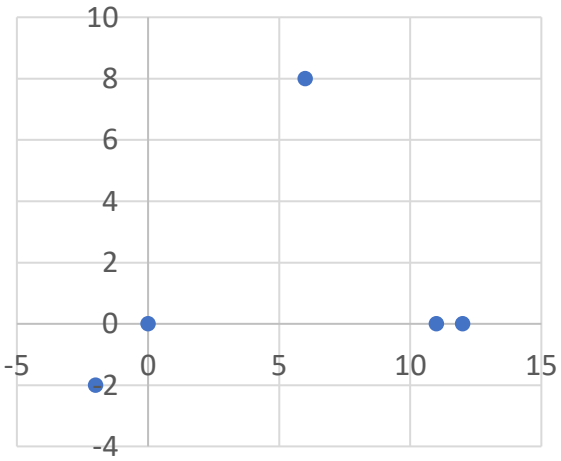
回答例を提供しています：shortestPath/dijkstra.c

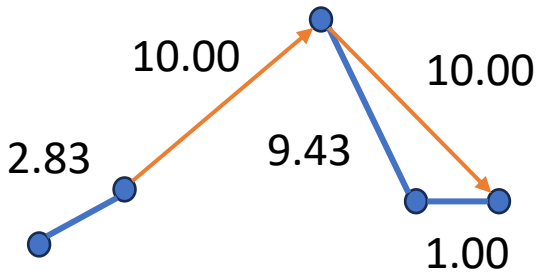
最短経路問題

盤面に点が配置されています.
始点から終点まで移動したいのですが,
２点間の距離が 10以下の場合だけ移動することができます.
以降で紹介する２種類のアルゴリズムを用いて，最短経路問題を解いてみましょう.

入力例

```
 0  0   スタート
 6  8
-2 -2
11  0
12  0   ゴール
```





課題４の紹介

- 2011年 ICPC 2010年アジア予選 福岡大会の問題Dをベースに

タクシーが長い距離を移動する。
都市の距離関係を示すグラフが与えられている
- タクシーは開始点からゴールに移動
- ガスステーションは少数
- 満タンにしても、指定した距離しか移動できない
  (cap x 10 しか移動できない, cap: タンクの容量)

質問：移動距離を最小にするには？(到達不能の場合は -1 を返す)

課題４の紹介

- 2011年 ICPC 2010年アジア予選 福岡大会の問題Dをベースに

入力例

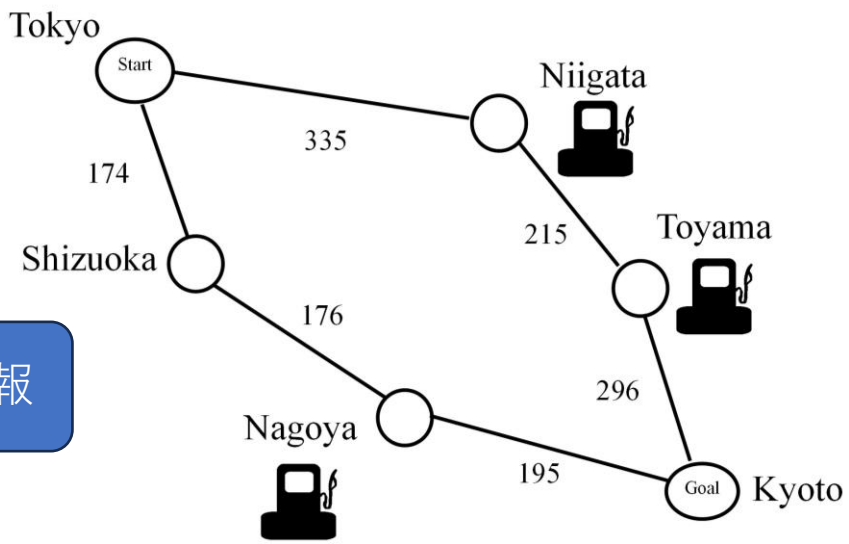
```
6 3 34
Tokyo       Kyoto
Tokyo       Niigata   335
Tokyo       Shizuoka  174
Shizuoka    Nagoya    176
Nagoya      Kyoto     195
Toyama      Niigata   215
Toyama      Kyoto     296
Nagoya
Niigata
Toyama
```

道路は６つ，ガソリンスタンドにある都市3つ，ガソリンタンクは34L

Tokyoから，Kyotoへ行きたい

道路の情報

ガソリンスタンドがある都市



課題４の紹介

- 課題４（基準課題）

本来の問題をもう少し簡単にする：

ガソリンタンクは無限であることを前提する

本来の問題は→加点課題対象

（次回もう少し詳しく説明する）

課題４の紹介

ソースコード：taxi/

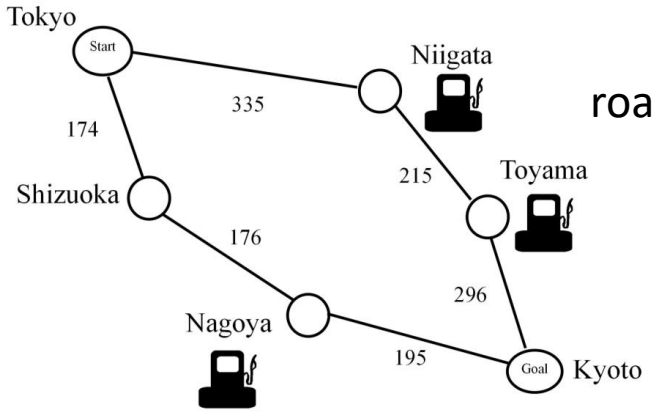
- longDistTaxi.c　　プログラムの枠組み
- Sample.in　　入力例
- Sample0.ans　　課題４の回答
- Sample.ans　　従来の問題の回答
- D.in/D0.ans/D.ans　同様

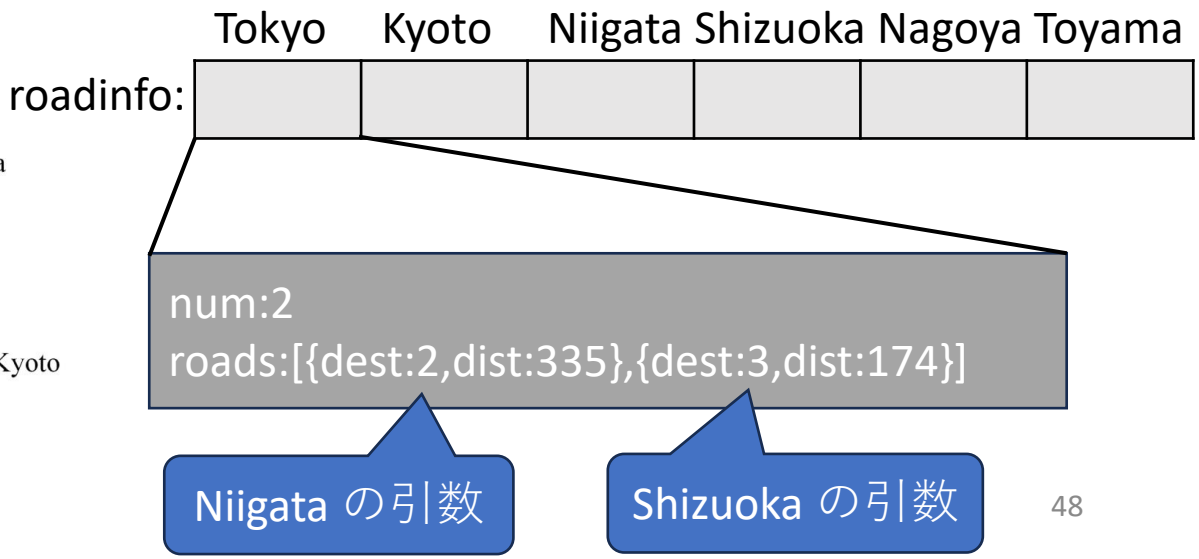
課題４の紹介

```
6 3 34
Tokyo       Kyoto
Tokyo       Niigata   335
Tokyo       Shizuoka  174
Shizuoka    Nagoya    176
Nagoya      Kyoto     195
Toyama      Niigata   215
Toyama      Kyoto     296
```

```
typedef struct roadInfo {
        int num;
        struct {
                int dest;
                int dist;
        } roads[MAX_N_ROADS];
} roadinfo_t;

/* 都市に関して，繋がっている道路 */
roadinfo_t roadinfo[MAX_N_CITIES];
```





課題 4

- 提出締め切り 2024年1月18日 12:00
  未完全でも必ず提出してください!
  - レポート
    - 実行結果（debug 出力が混ざっていても構いません）
    - 課題 4 は、追加したデータ構造などの簡単な説明を入れてください
    - ほとんどのデータで正解しているが、一部データでバグが取れない場合は、その旨記述して、考察を記載してください
    - アピールポイントがある人は、その旨解説入れてください
  - ソースコード
- 回答例を1月18日に公開します
- 再提出締め切りは1月25日 12:00
  - 自分が作成できなかった部分がどの部分か、正解例ではどのように実現されていたか、解説すること
  - 正解プログラム例の実行を、デバッガなどでトレースし、正しく動作していることを説明すること．今回の例では，sample.in の最初の例について解説すれば OK です．

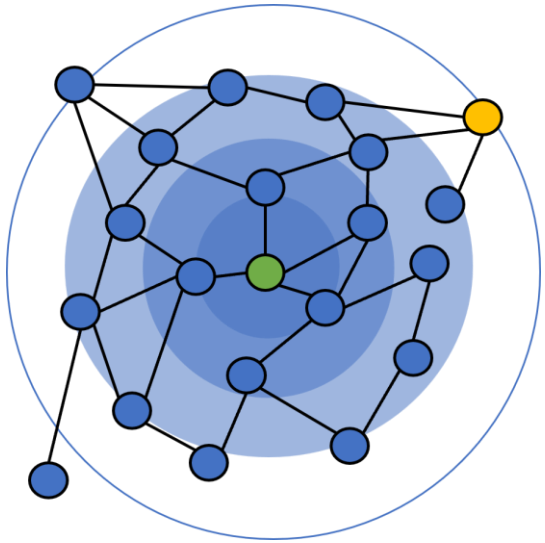
８回目　内容

- **A＊紹介**
  - **ダイクストラ法の欠点**
  - **優先度，ダイクストラ法とA＊比較**
  - **実装例**
  - **最適性**

- **加点課題説明**
  - **課題６**
  - **課題５**

ダイクストラ法の欠点

- スタートの周りに全方向で上々に探索する
- ゴールを決まった時に逆方向も無駄に探索する



- ゴールに**近づく**ようなノードを**優先的に探索**できないでしょうか？

A＊アルゴリズム

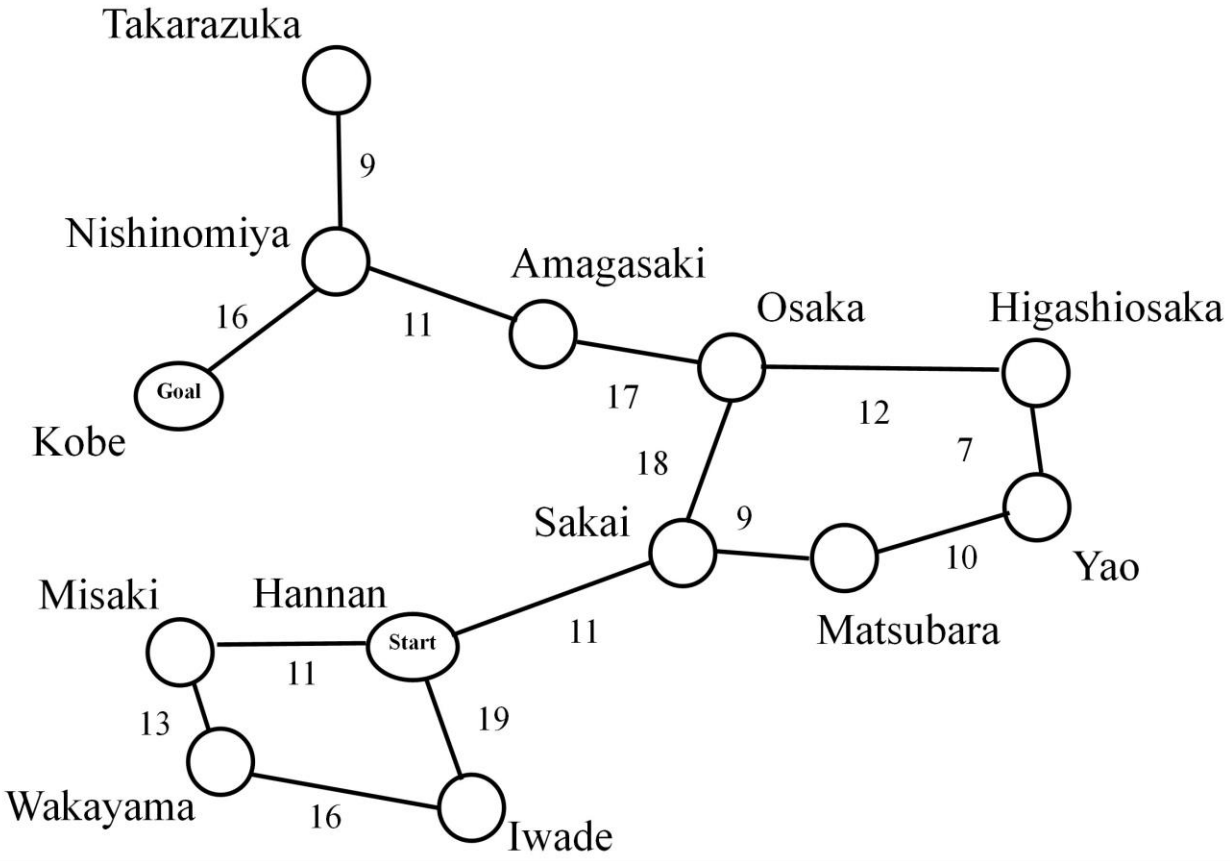
- ダイクストラ法と全く同じアルゴリズムです
- 違うところはただ１つ：**優先度の求め方**

ダイクストラ法：
当ノードまでかかった距離

A＊：
当ノードまでかかった距離
＋ペナルティ

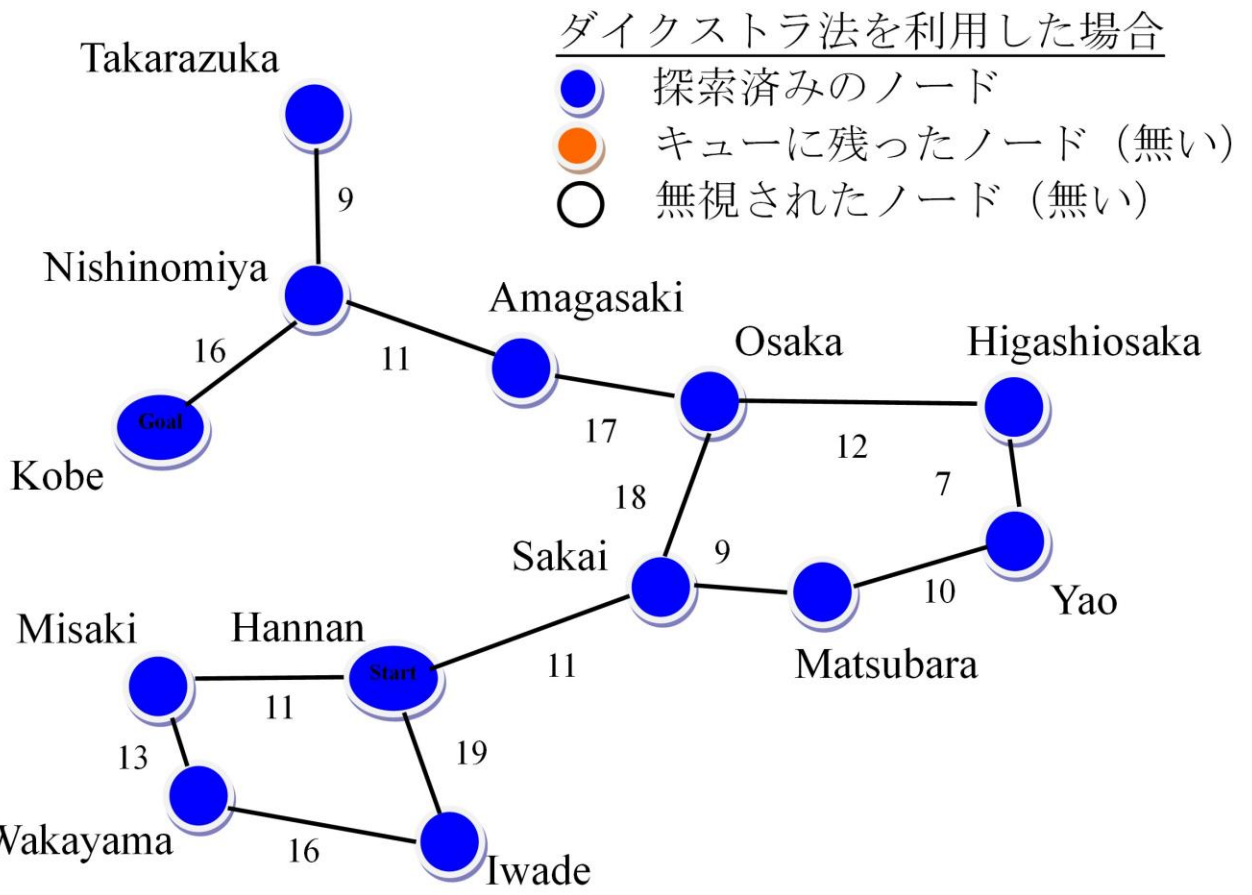
- **このペナルティは当ノードからゴールまでかかりそうな距離（予測）**
- その影響で，ゴールに近づくノードが優先度が高くなり，早めにキューから取り出される

A＊のメリット　具体例

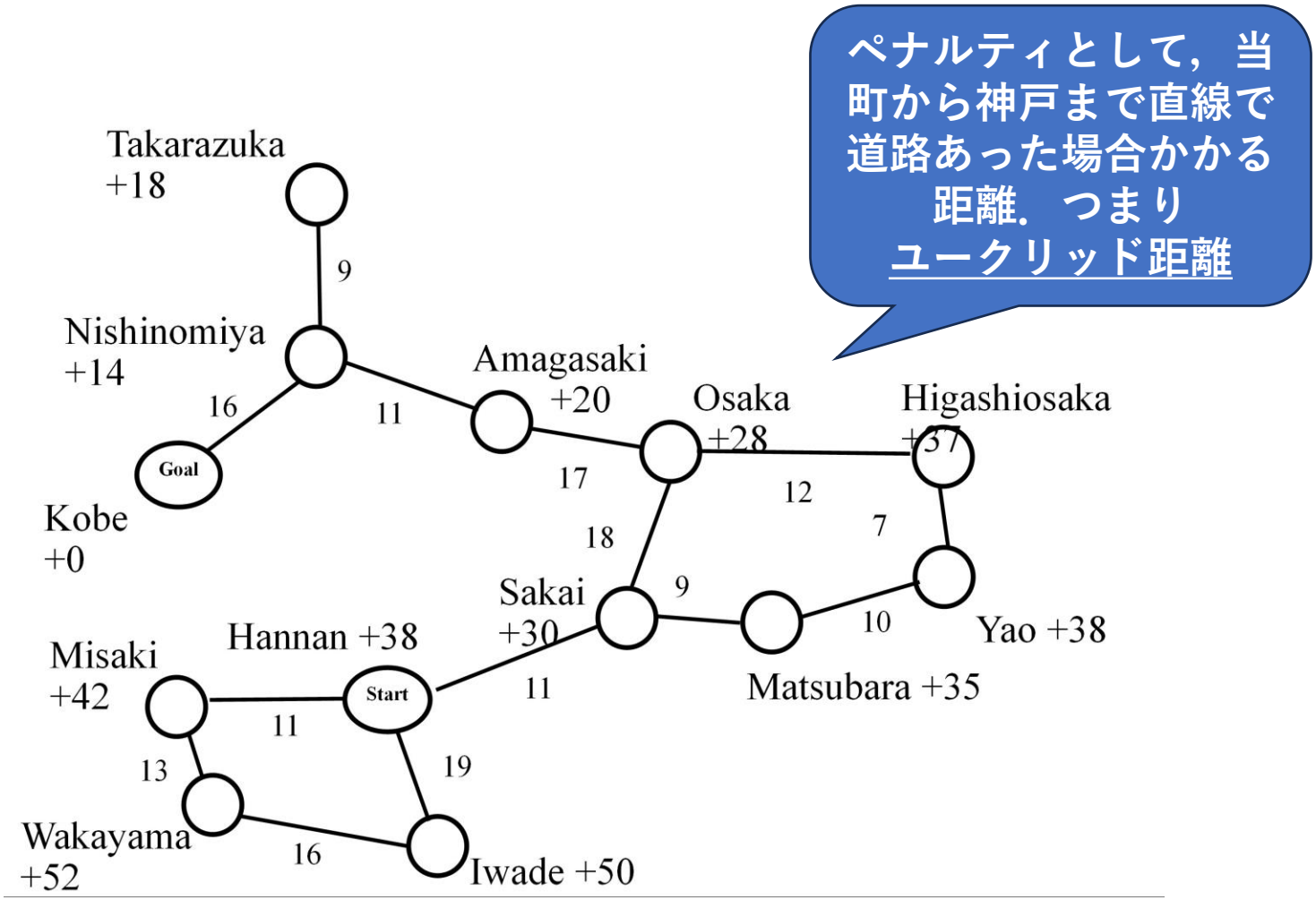

Takarazuka
9
Nishinomiya
Amagasaki
16
11
Osaka
Higashiosaka
Goal
17
12
Kobe
18
7
Sakai
9
10
Yao
Misaki
Hannan
Start
11
Matsubara
11
13
19
Wakayama
16
Iwade

A＊のメリット　具体例

ダイクストラ法を使った場合

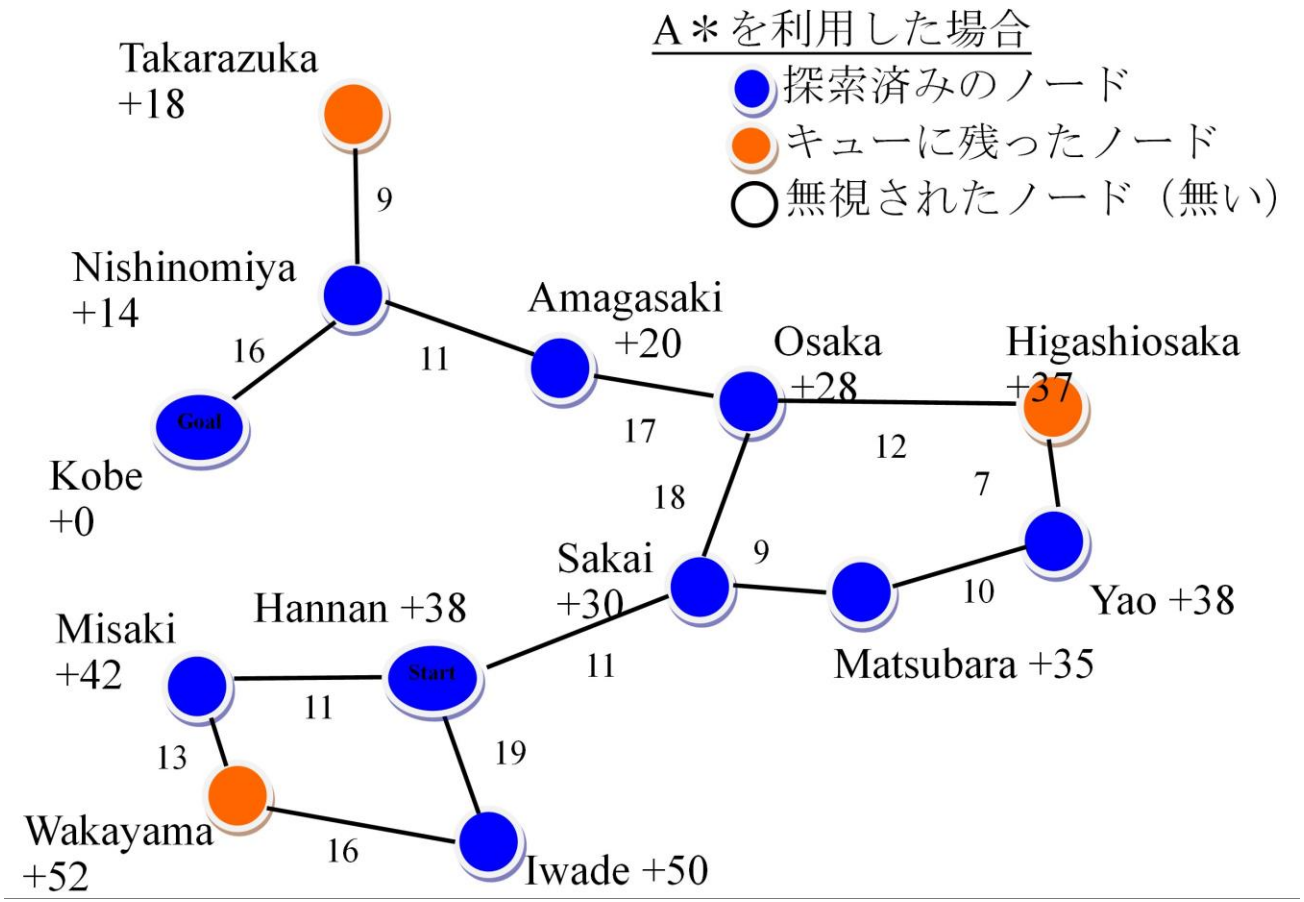


A＊のメリット　具体例


ペナルティとして，当町から神戸まで直線で道路あった場合かかる距離．つまり
ユークリッド距離
Takarazuka
+18
9
Nishinomiya
+14
16
11
Amagasaki
+20
Osaka
+28
Higashiosaka
+37
Goal
17
12
Kobe
+0
18
7
Sakai
+30
9
10
Yao +38
Hannan +38
Misaki
+42
Start
11
Matsubara +35
11
13
19
Wakayama
+52
16
Iwade +50

A＊のメリット　具体例

A＊を使った場合



ダイクストラ法とA＊の比較

- 下記の動画をご覧ください！

https://www.youtube.com/watch?v=BR4_SrTWbMw

A＊の最適性

- 探索順を変えて，得られる解は本当に最短ルートなのか？

ペナルティは任意のノードからゴールまでの**過小評価**でしたら，最短ルートを確実に得られる

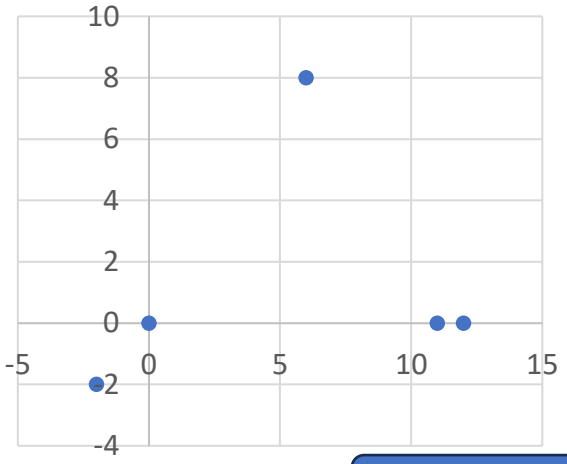
そうでない場合は，探索は更に早くなりますが，得られる解は実際に最短であるかは保証がない

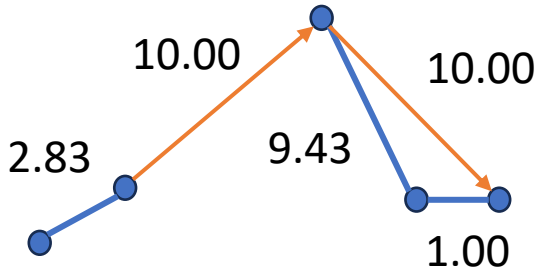
実装

A＊の実装例を提供しています：
shortestPath/astar.c

入力例

```
 0  0   スタート
 6  8
-2 -2
11  0
12  0   ゴール
```





要素をキューに導入するとき（l198~201）

ノードjまでかかる距離

ノードjからgoalまでかかる距離 (予測)

```
cost[j] = nextPathLen;
double est = distance(points[j], goal);
searchNode_t next = { nextPathLen, nextPathLen + est, j};
enqueue(&Q, next);
```

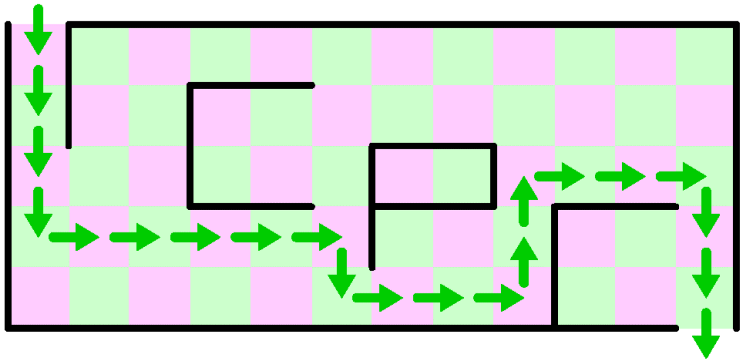
優先度

```
typedef struct searchNode {
  double pathLen;
  double plusEstimation;
  int index;
} searchNode_t;
```

```
int compare(searchNode_t * a, searchNode_t * b) {
  if(a->plusEstimation < b->plusEstimation) { return -1; }
  if(a->plusEstimation == b->plusEstimation) { return 0; }
  return 1;
}
```

課題６

- もう一度課題２を観てみましょう



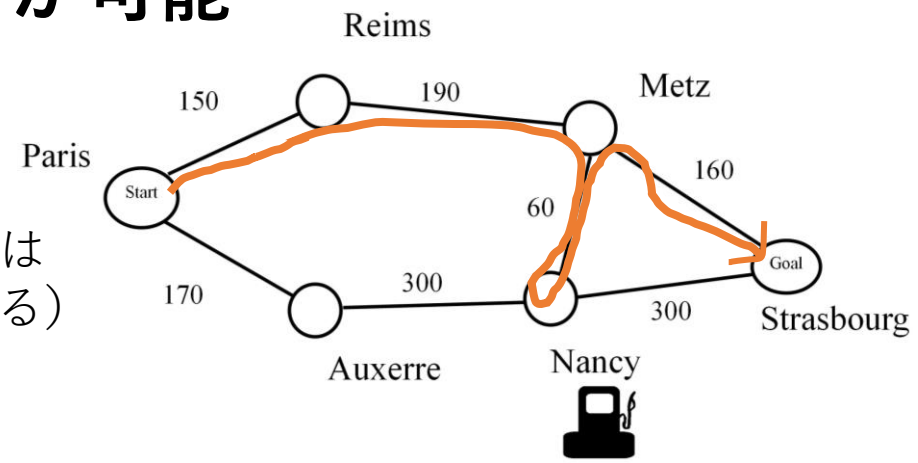
- 幅優先探索で解けたのですが，A＊を使うもっと効率的な実装が出来そう
- A＊を実装して，幅優先探索と性能を比較してみてください．詳細や検討事項などはウエブページにご確認ください

課題５　補足

ダイクストラ法をベースに，ノードをキューに加える条件がややこしくなった版

- **解け方１**

通常のダイクストラ法だと，あるノードを一度訪問したら，２度と訪問しない．
ガソリンタンクの容量を配慮した場合は，**ガソリンスタンドのない町を複数回探索**することが許さないといけない．理由は：**以前届けない町にたどり着くことが可能**



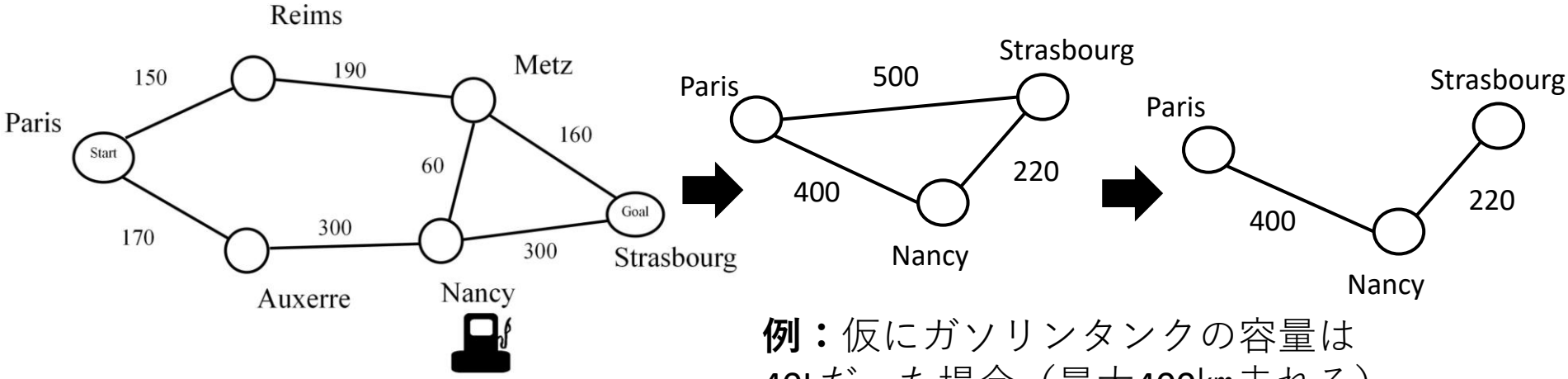
例：仮にガソリンタンクの容量は40Lだった場合（最大400㎞走れる）

課題５　補足

実装はもう少し長いですが，通常のダイクストラ法を使いますので，比較的にデバッグしやすい版

- **解け方２**

ダイクストラ法を使って，**ガソリンスタンドありの町**しかない**の全結合グラフを作成**して，ガソリンタンク容量を配慮し，**走れない枝を削除**して，**通常のダイクストラ法で解ける**



例：仮にガソリンタンクの容量は40Lだった場合（最大400㎞走れる）

実施アンケート

Beef＋へ

ご協力をお願いします