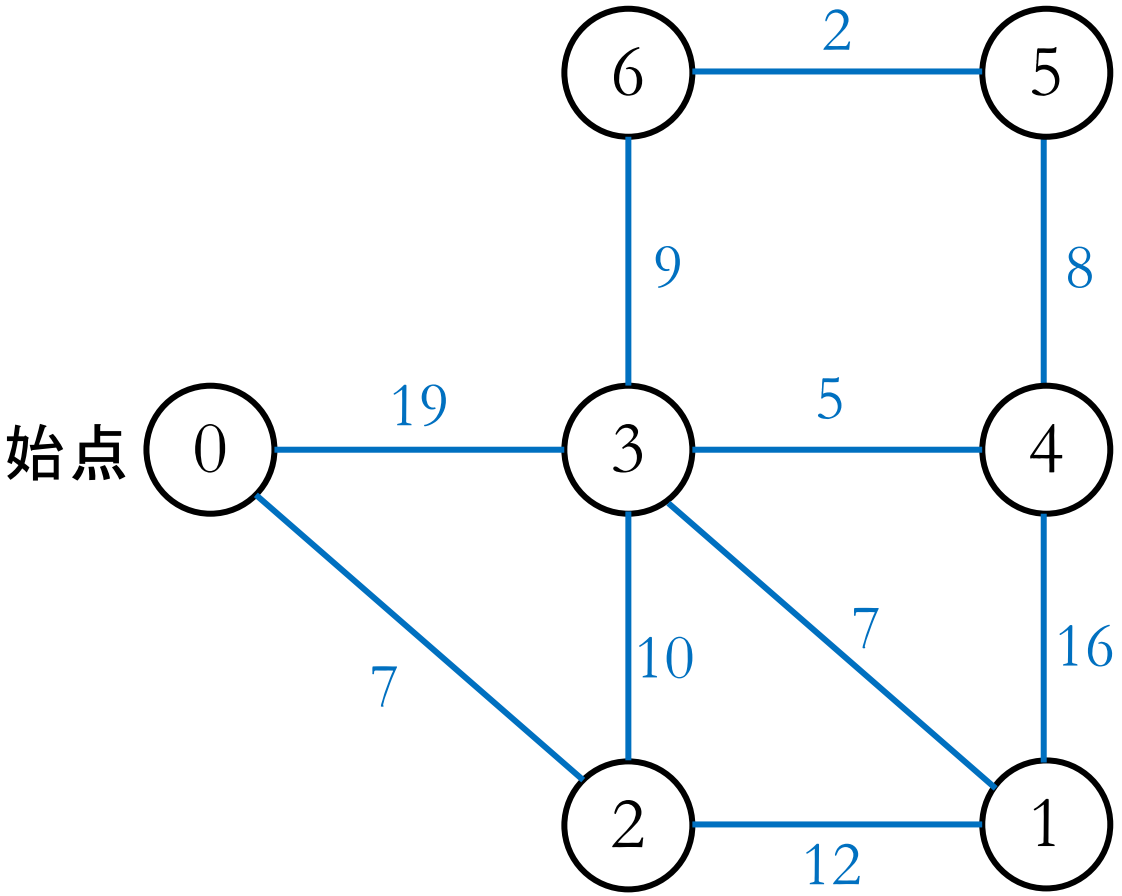
動的計画法（DP）

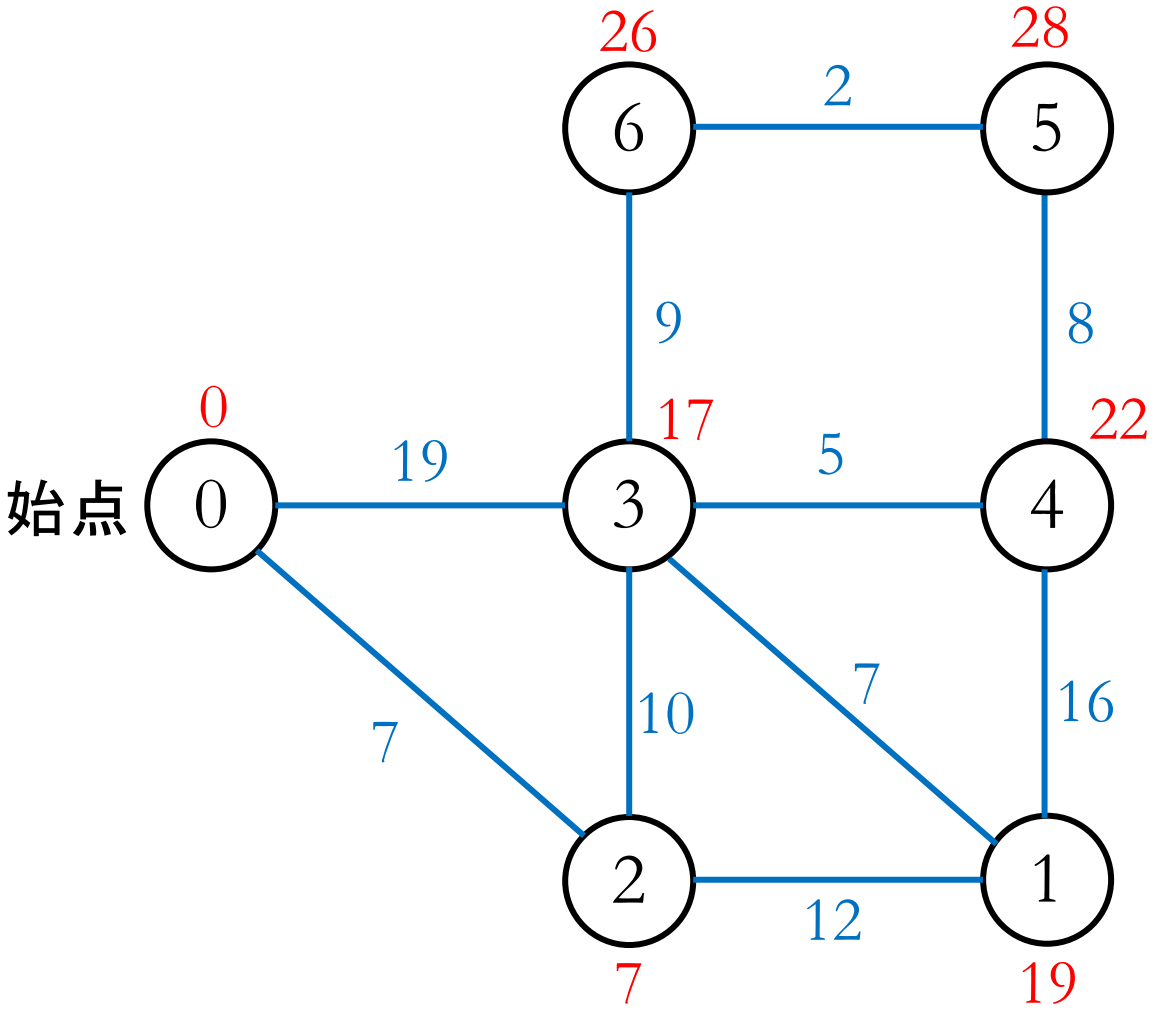
最短経路

最短経路問題

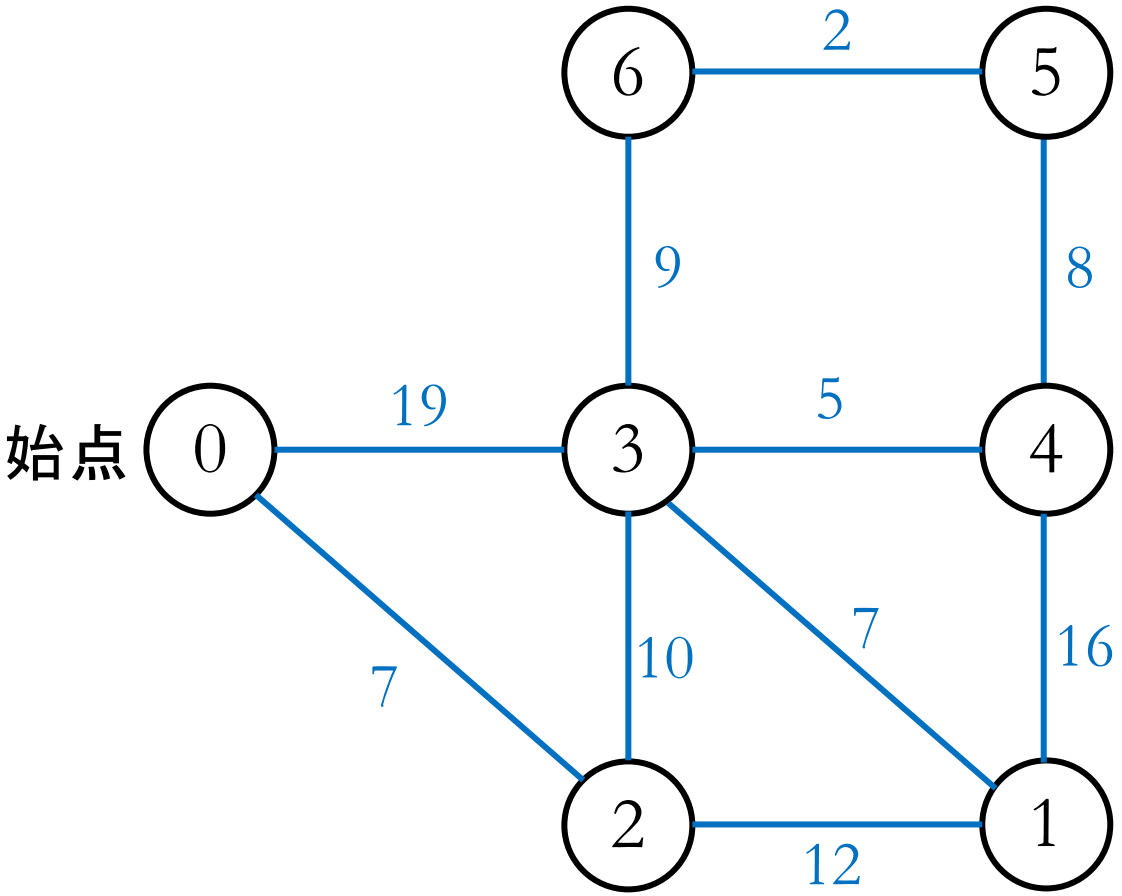
- 重み付きグラフ $G=(V, E)$ が与えられた時に、任意の2頂点を結ぶ経路の中から、辺の重みの総和が最小のものを求める問題
- カーナビで今の場所から目的地への最短の道順の探索などに利用
- 重みは距離・時間やコストなどに言い換えることができる。ここの最短経路問題においては距離という言葉を使う

最短経路の例

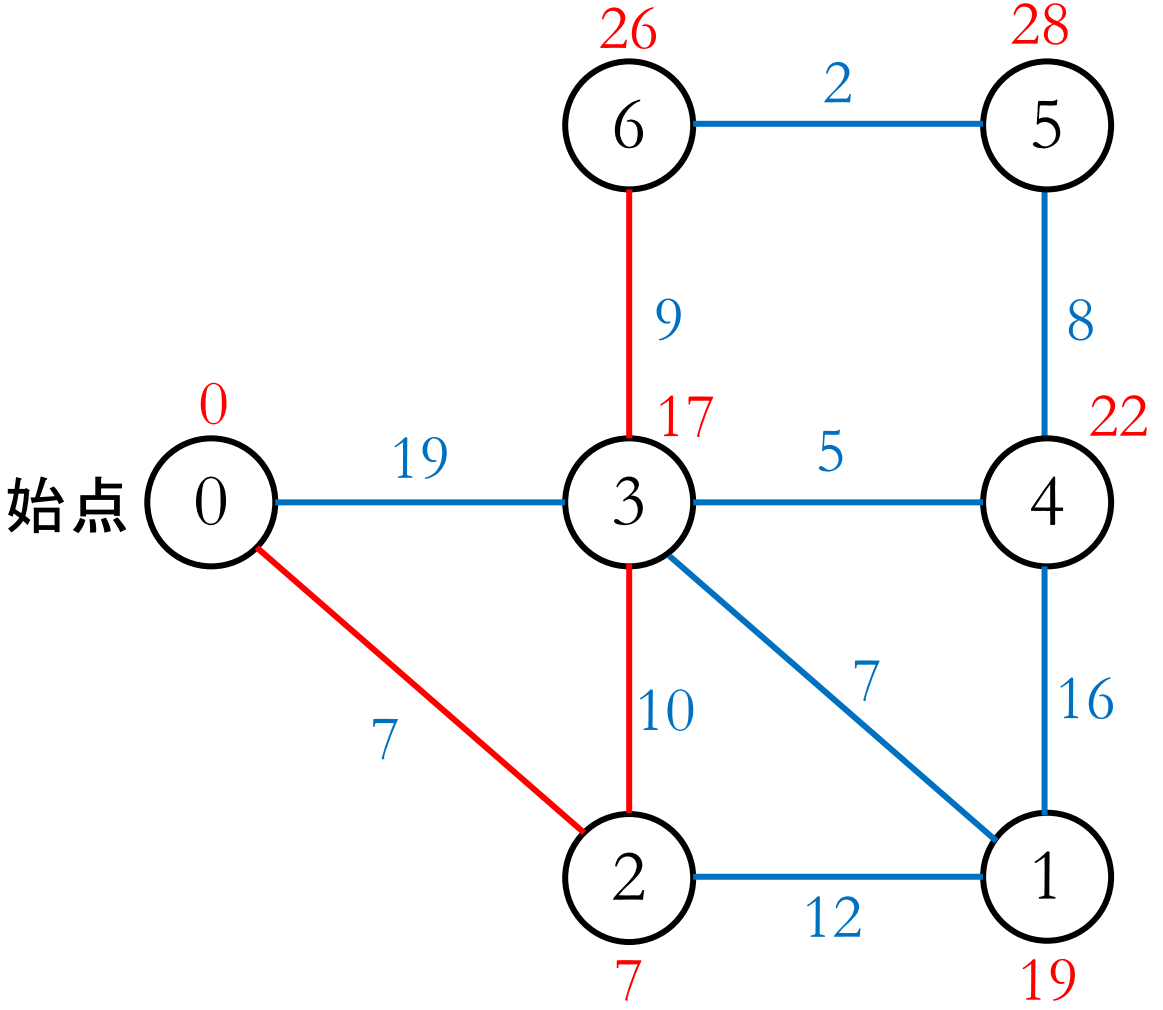

始点
0
19
3
5
4
6
2
5
9
8
7
10
7
16
2
12
1


始点
0
0
19
3
17
5
4
22
26
6
2
28
5
9
8
7
10
7
16
2
12
1
7
19

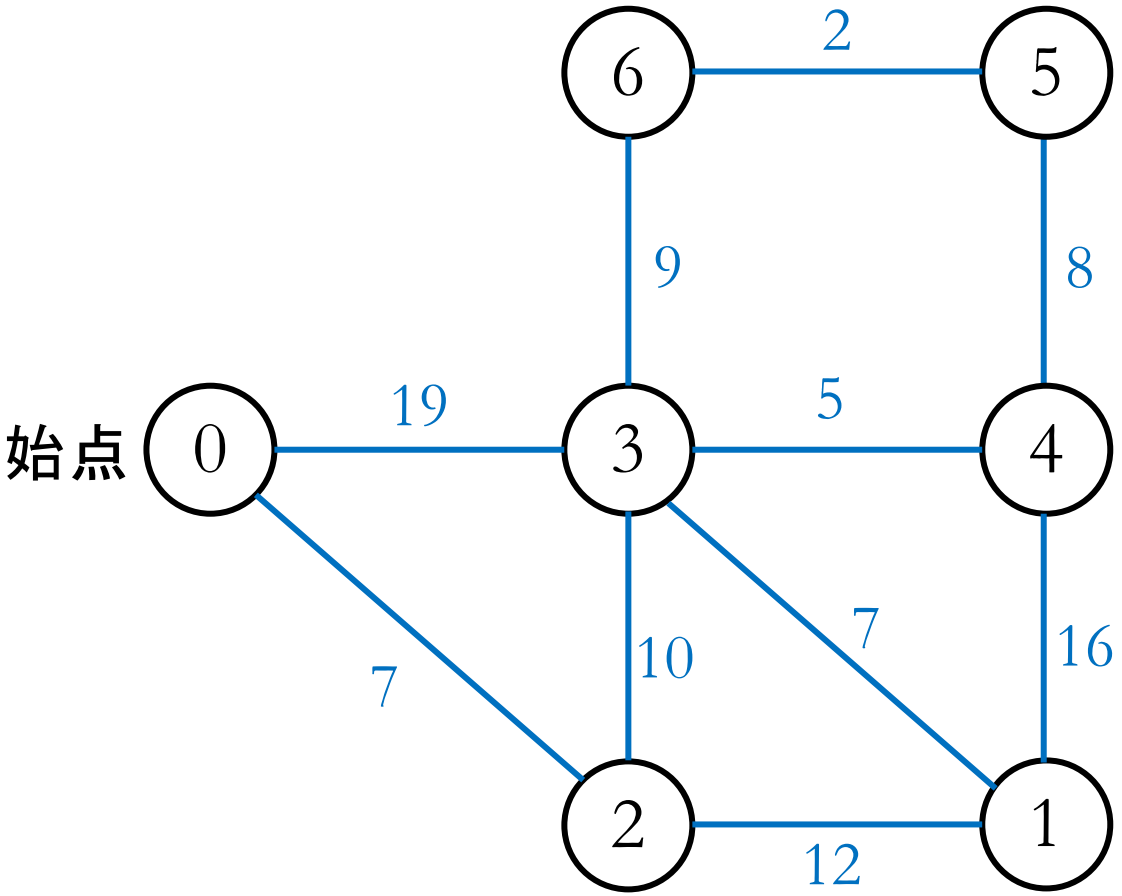
最短経路の例



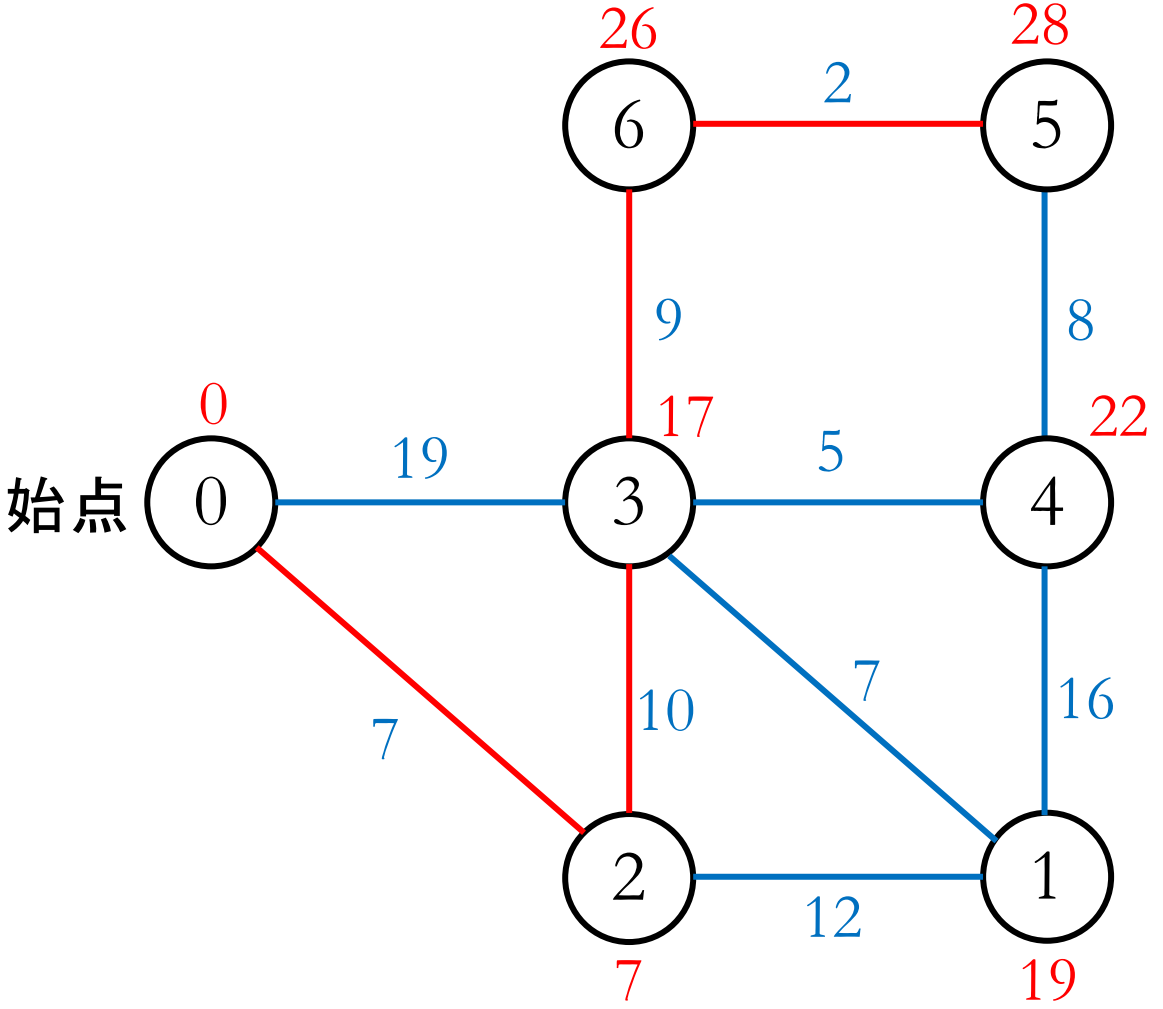
頂点0→6の場合



最短経路の例



頂点0→5の場合



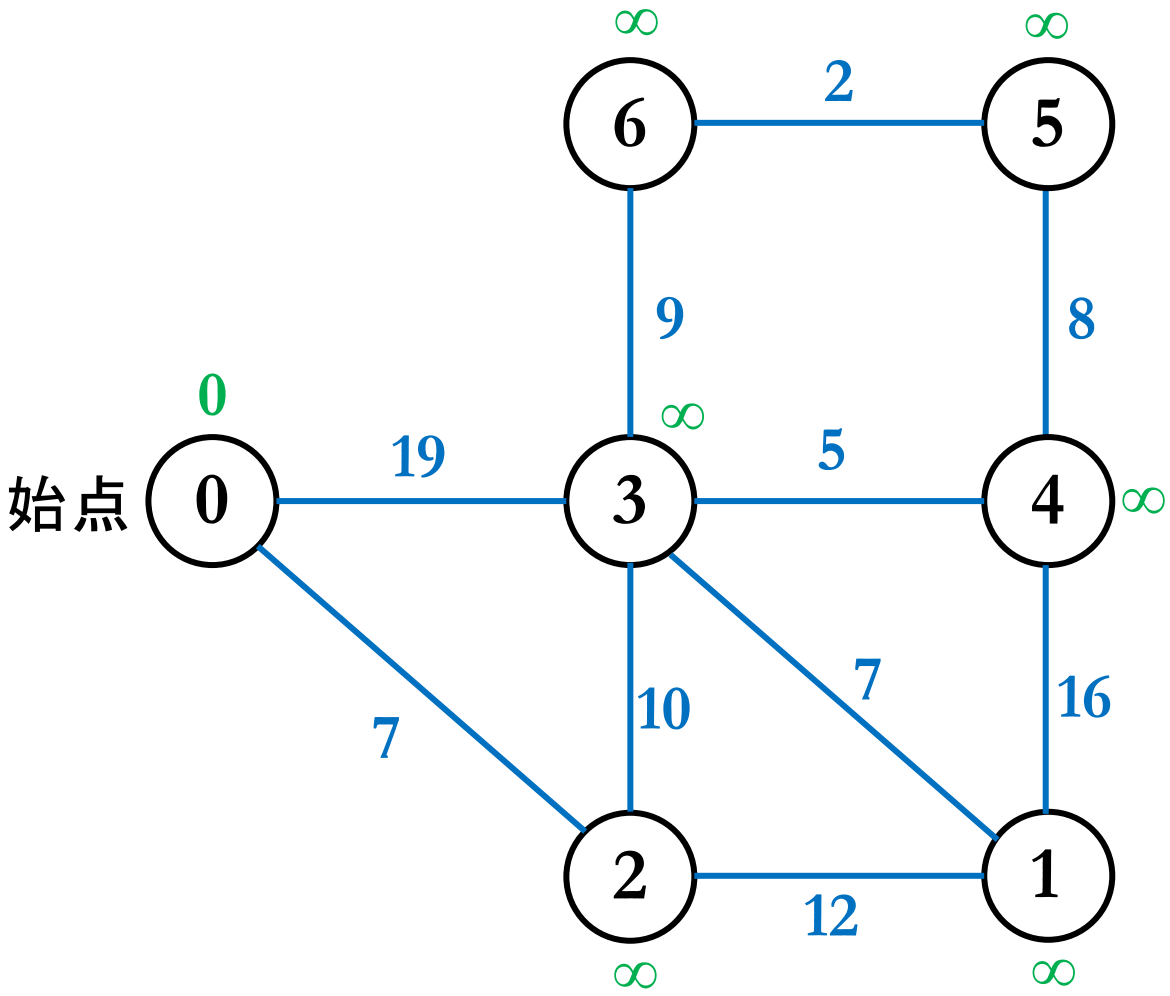
ダイクストラ法（Dijkstra's algorithm）

- 特定の始点からすべての頂点への最短経路（距離）を同時に求める方法
- 始点から隣接関係を用いて幅優先的に最短距離の頂点を決め、その頂点からさらに最短な距離の頂点を決めていく
- その段階的に各段の結果を記憶・利用しながらすべての頂点をもとめる方法はまさに多段決定問題である

ダイクストラ法の例

- グラフを表現する隣接行列

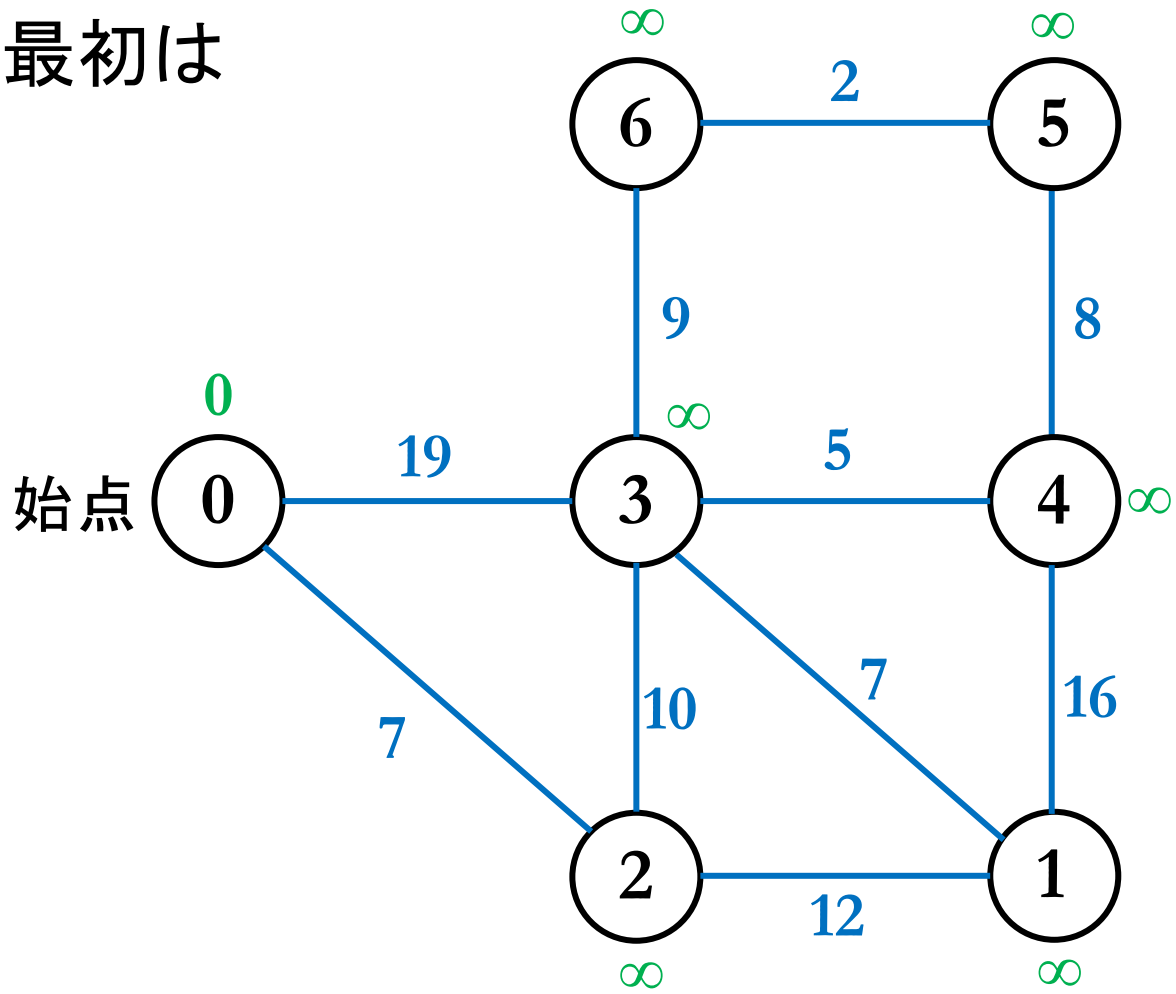
	0	1	2	3	4	5	6
0	0	∞	7	19	∞	∞	∞
1	∞	0	12	7	16	∞	∞
2	7	12	0	10	∞	∞	∞
3	19	7	10	0	5	∞	9
4	∞	16	∞	5	0	8	∞
5	∞	∞	∞	∞	8	0	2
6	∞	∞	∞	9	∞	2	0



ダイクストラ法の例

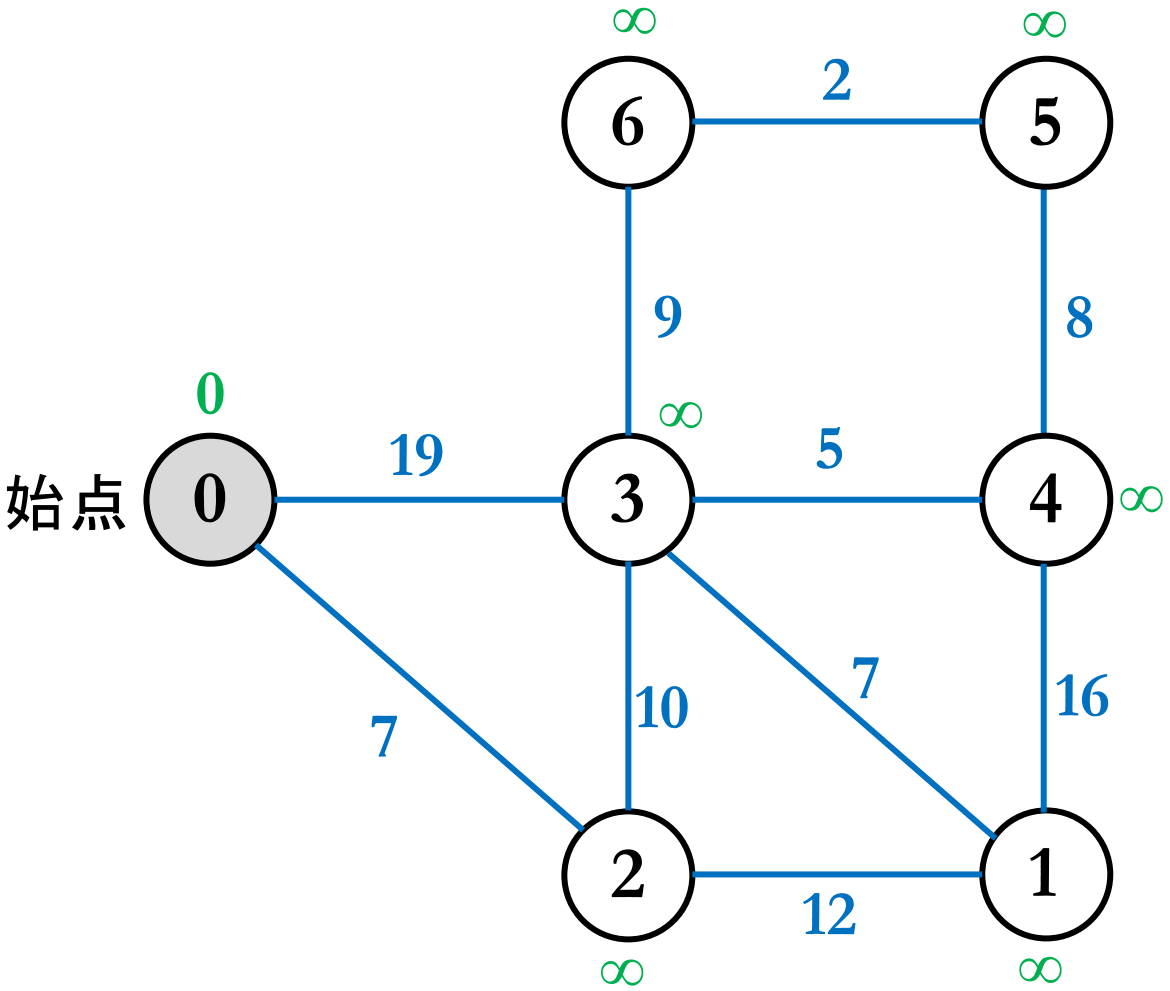
- 始点から頂点の距離は始点以外最初は無限大

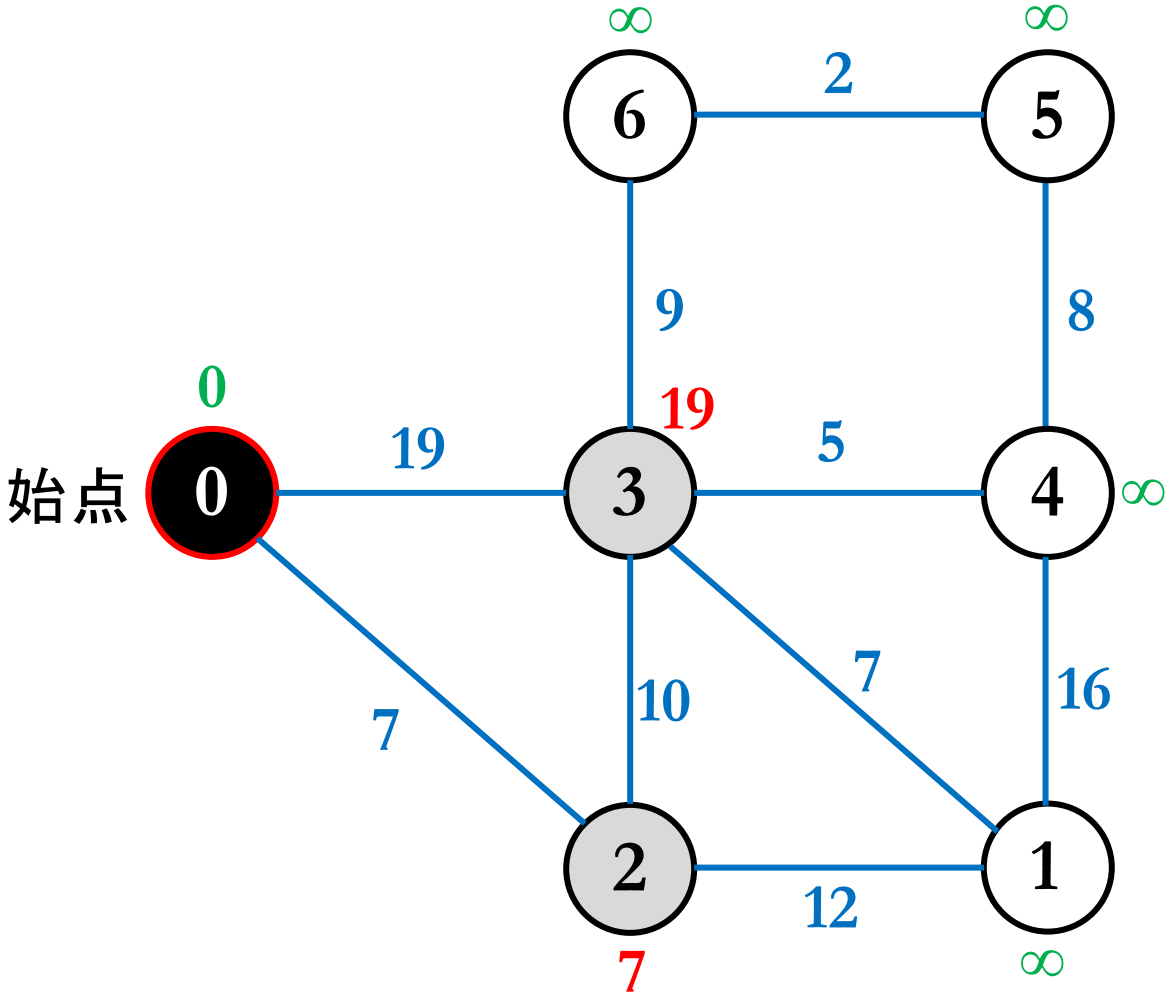
0	1	2	3	4	5	6
0	∞	∞	∞	∞	∞	∞



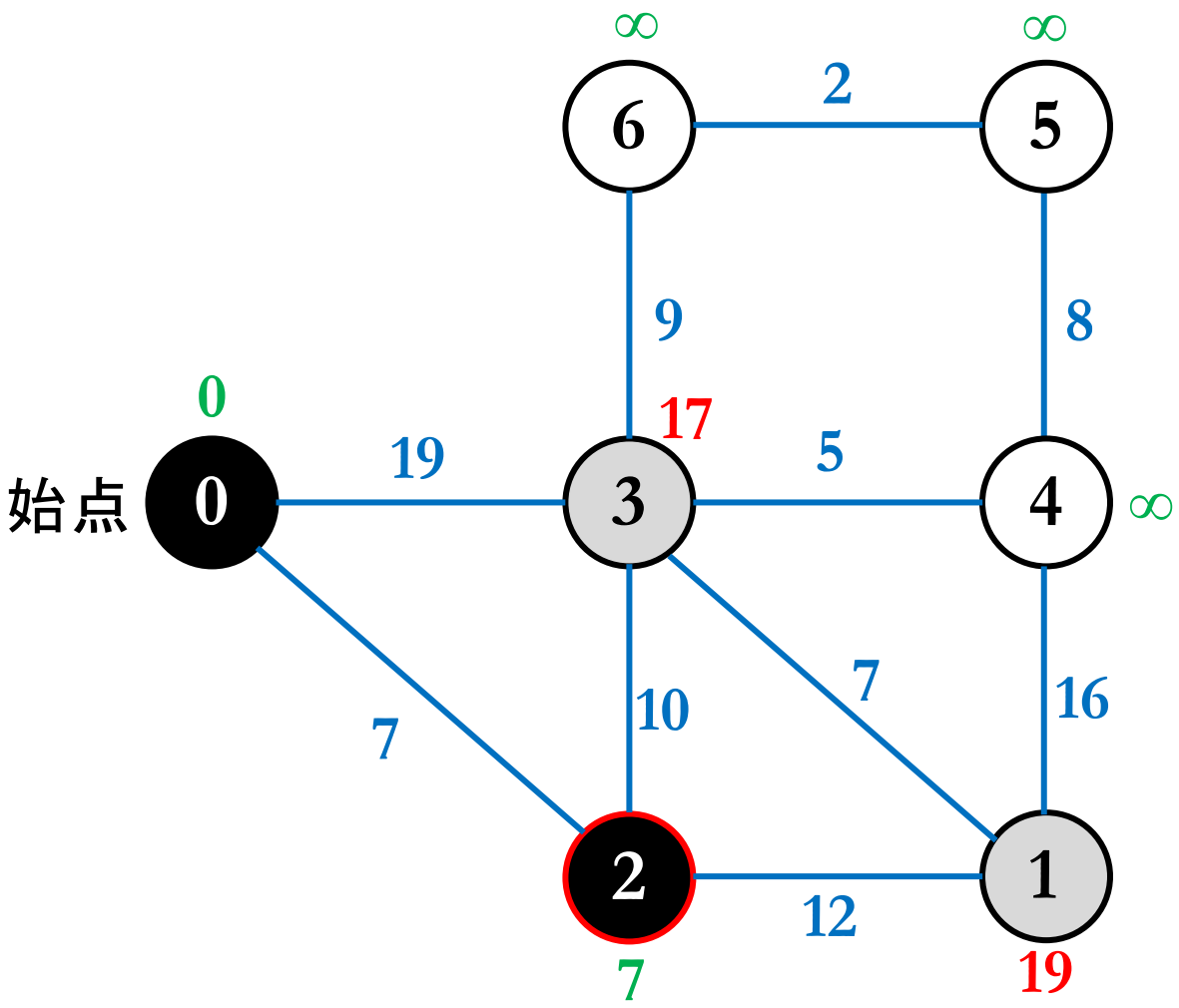
- 白い頂点：未処理
- グレイ頂点：解候補（処理中）
- 黒い頂点：解（処理済み）

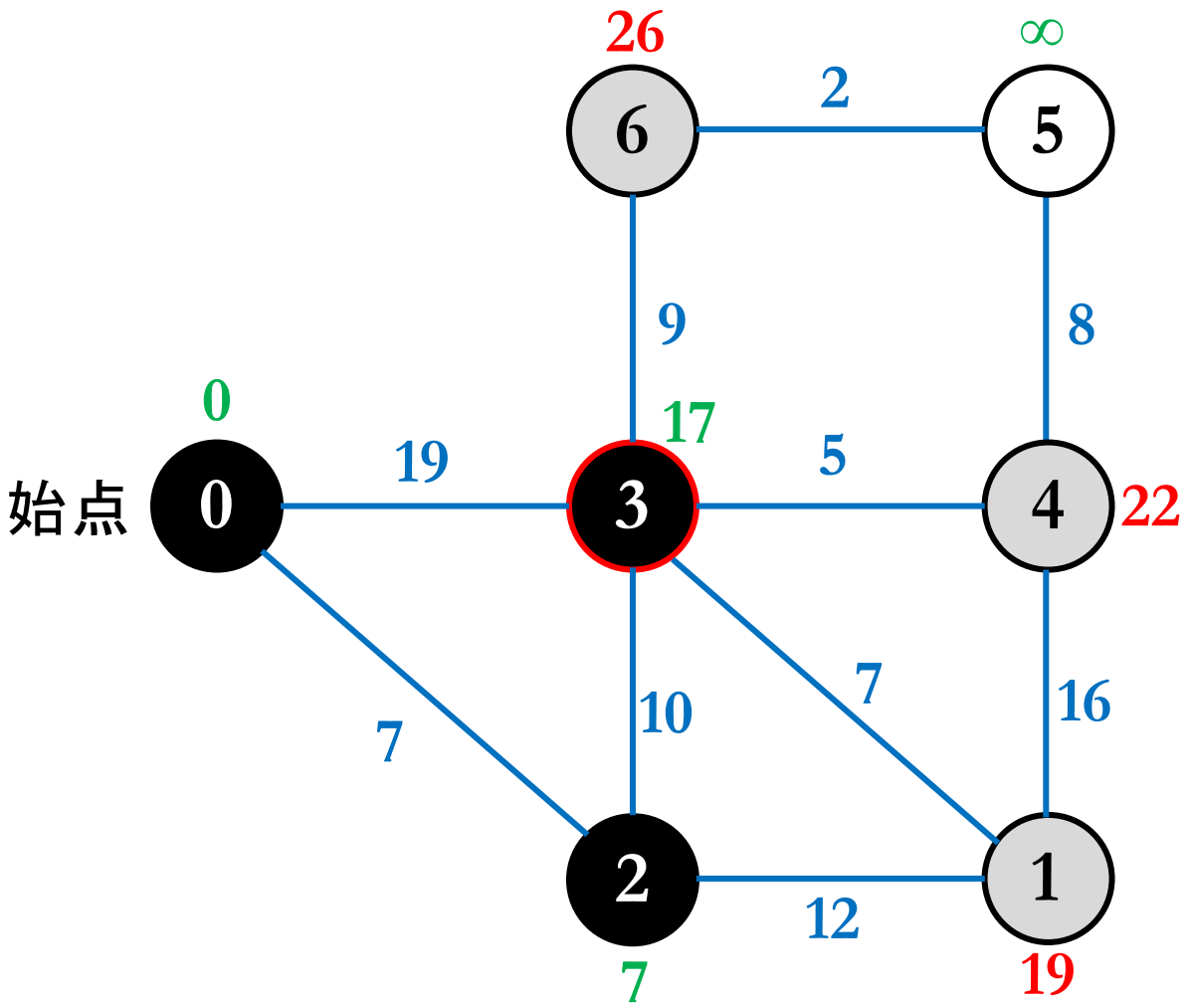
ダイクストラ法の例


∞
∞
6
2
5
9
8
0
∞
始点
19
5
∞
0
3
4
7
10
7
16
2
1
12
∞
∞

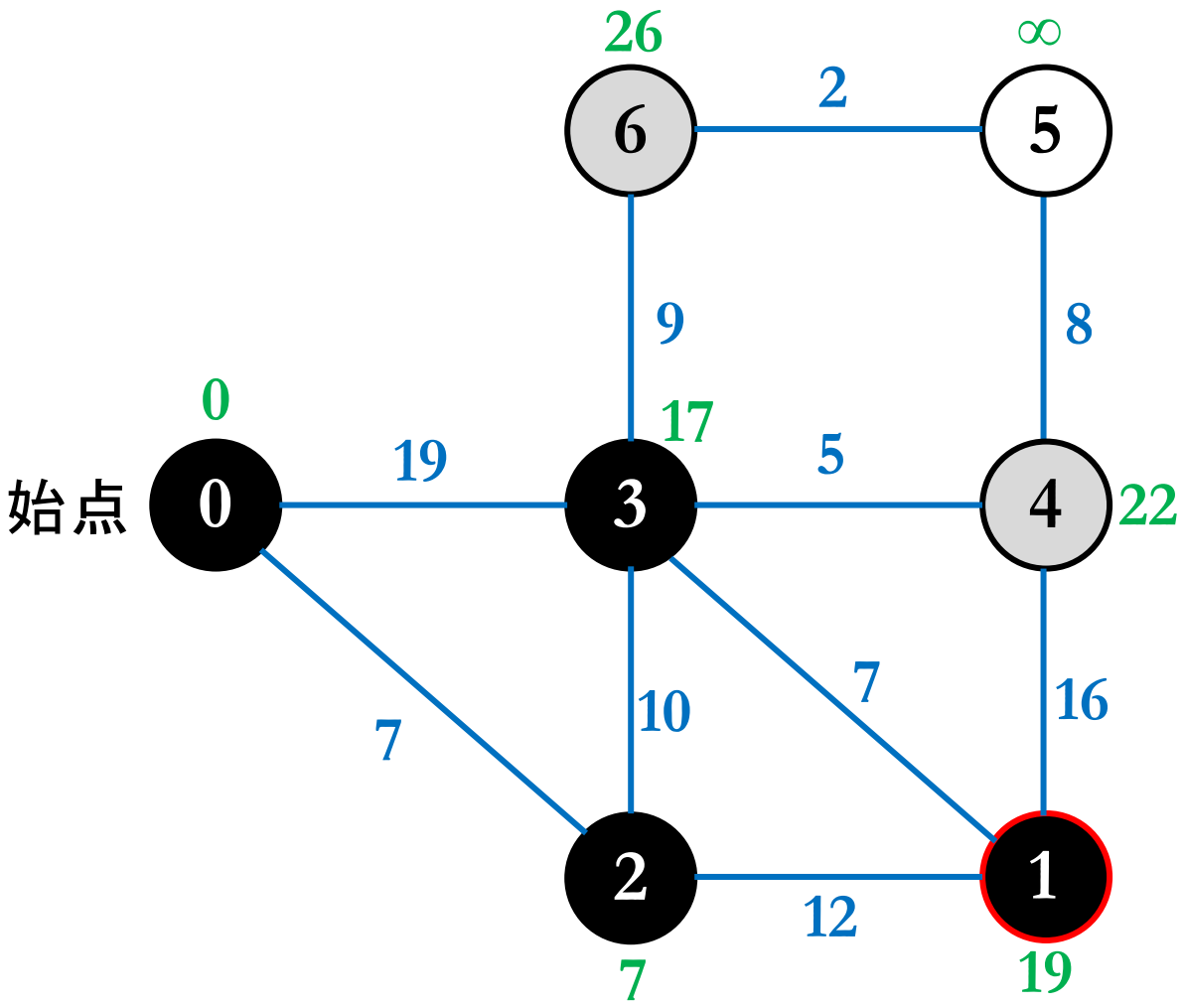

∞
∞
6
2
5
9
8
0
19
始点
19
5
∞
0
3
4
7
10
7
16
2
1
12
7
∞

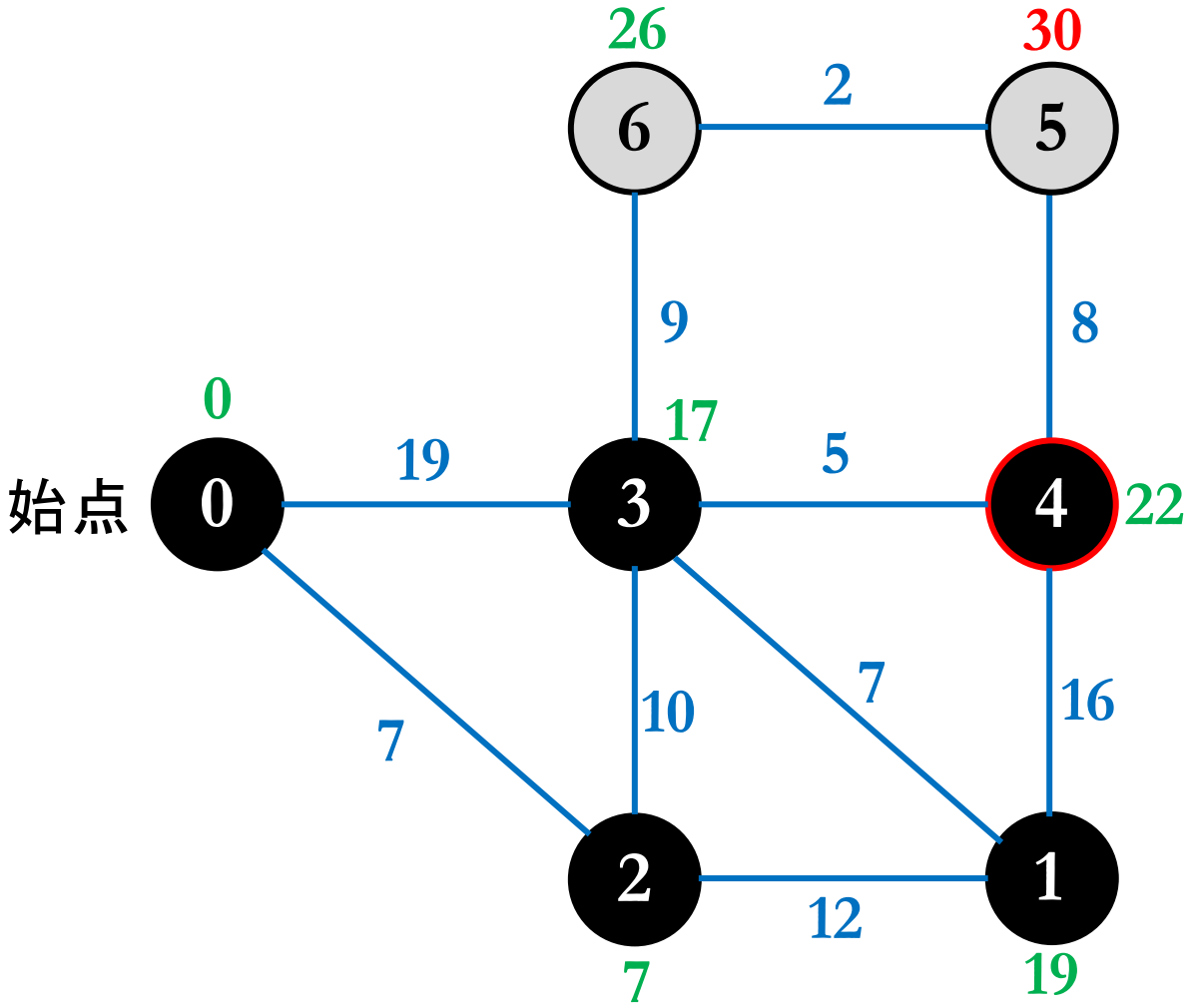
ダイクストラ法の例


∞
∞
6
2
5
9
8
0
17
始点
0
19
3
5
4
∞
7
10
7
16
2
12
1
7
19

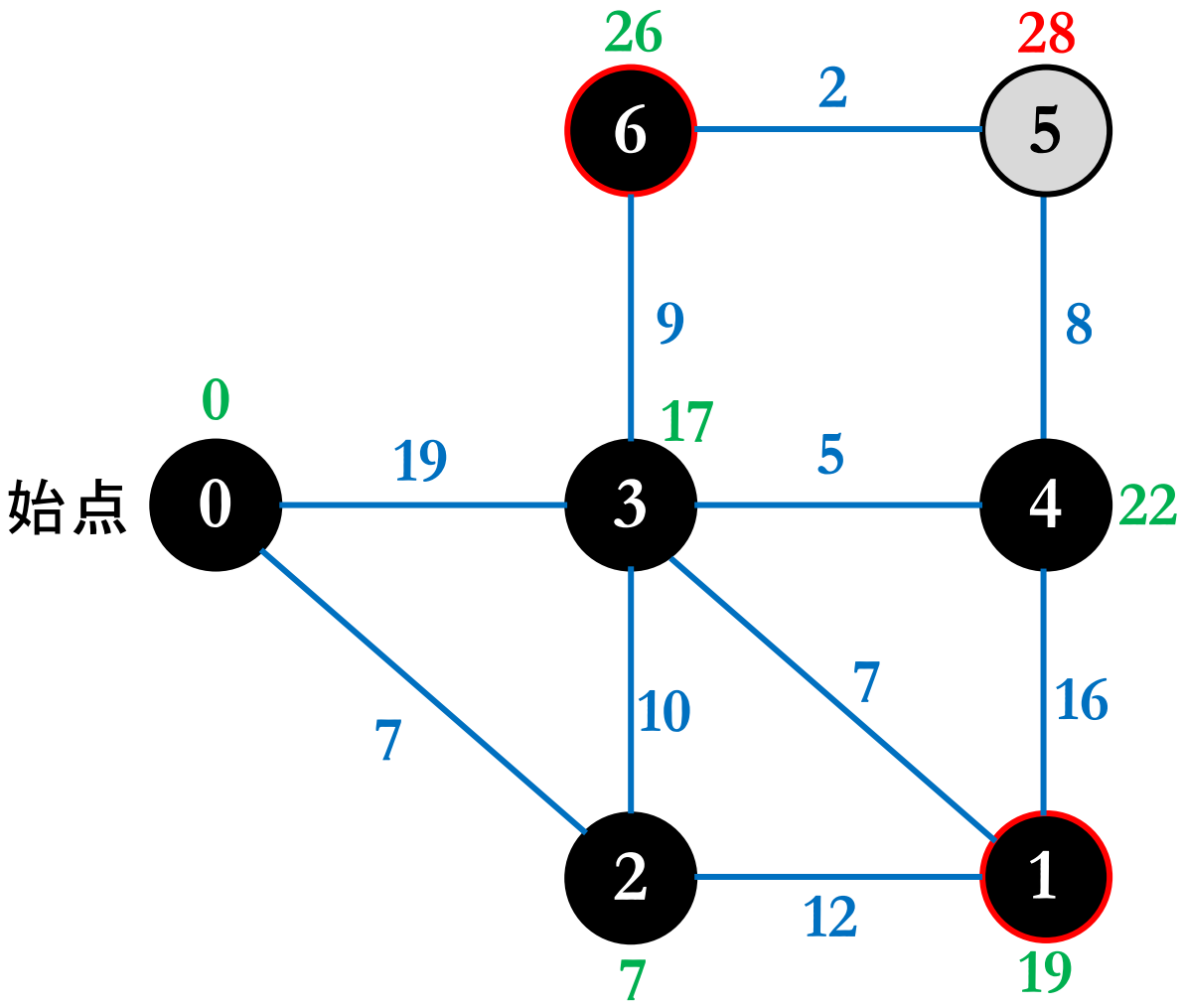

26
∞
6
2
5
9
8
0
17
始点
0
19
3
5
4
22
7
10
7
16
2
12
1
7
19

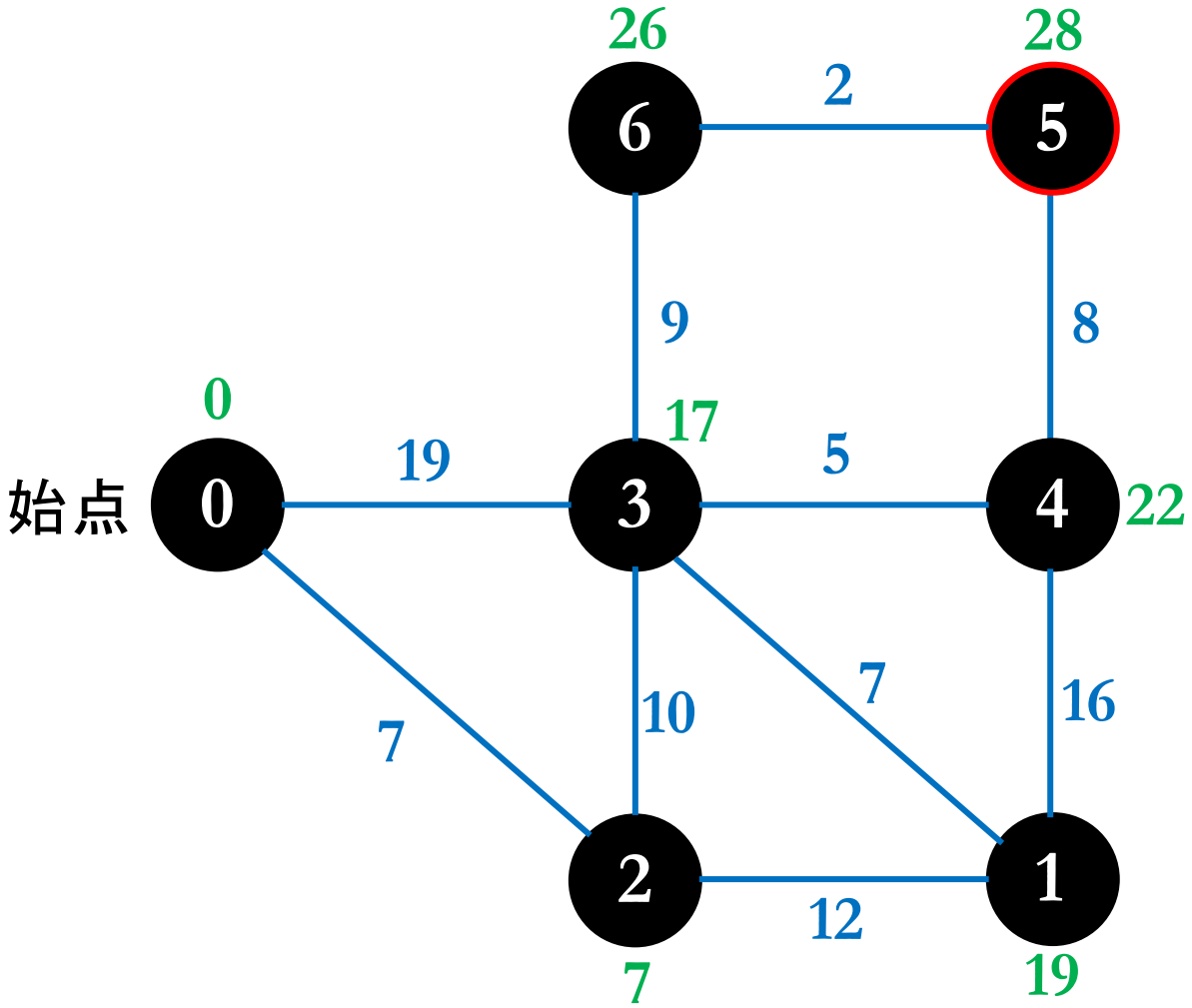
ダイクストラ法の例


始点
0
0
19
3
17
5
4
22
6
26
2
5
∞
9
8
7
10
7
16
2
12
1
7
19


始点
0
0
19
3
17
5
4
22
6
26
2
5
30
9
8
7
10
7
16
2
12
1
7
19

ダイクストラ法の例


26
28
6
2
5
9
8
0
17
始点
0
19
3
5
4
22
7
10
7
16
2
1
12
7
19


26
28
6
2
5
9
8
0
17
始点
0
19
3
5
4
22
7
10
7
16
2
1
12
7
19

アルゴリズム

1. 初期設定
   1.1 各頂点を未処理のものとする
   1.2 各頂点に始点からの距離値∞（十分大きい値）を与える
   1.3 始点startに（始点からの）距離値0を与える
   1.4 startを解候補とする
2. 以下を繰り返す
   2.1 解候補から距離値の最小のもの解sを見つける
   2.2 解が見つからなければ終了
   2.3 sを解とし、sと隣接する頂点で解となっていないものiを解候補とし、iについて「頂点sの距離値+(s, i)間の距離値」を計算し、iの距離値を小さい方で更新する

manaba小テスト：06-1

- 0から各頂点への最短距離を
  求めなさい
- 15分
- 12点

実装にあたって

- 「未処理」「解候補」「解」は、ラベルを付けるという方法で実現できる
  - たとえば、未処理を「白」、解候補を「グレイ」、解を「黒」
- 「解が見つかるかどうか」の判定。一番思いつきやすい方法は解の数を数える方法。もう一つは、解（となる頂点）の番号を-1などあり得ない値で初期化し、解がなければ解の番号が-1のままになるはず、というような方法

実装にあたって

- 無限大と色を以下のように定義して使うと便利:

```
#define Inf 1000000
#define White 0
#define Gray 1
#define Black 2
```

- 各頂点を以下のような構造体で定義して使うと便利

```
typedef struct{
  int c, d, h;      // c: 色   d: 距離値
                    // h: 前の頂点を記憶用(経路そのものを出力するため)
}NODE;
```

アルゴリズムの実装

```
#include <stdio.h>

#define N 7 // 頂点の数

#define Inf 1000000
#define White 0
#define Gray 1
#define Black 2

typedef struct{
  int c, d, h;
}NODE;
```

```
NODE node[N];

int a[N][N] = {
  {0, Inf, 7, 19, Inf, Inf, Inf},
  {Inf, 0, 12, 7, 16, Inf, Inf},
  {7, 12, 0, 10, Inf, Inf, Inf},
  {19, 7, 10, 0, 5, Inf, 9},
  {Inf, 16, Inf, 5, 0, 8, Inf},
  {Inf, Inf, Inf, Inf, 8, 0, 2},
  {Inf, Inf, Inf, 9, Inf, 2, 0}
};
```

アルゴリズムの実装

```
void Dijkstra(int start)
{
 int i, dmin, s;

 // グラフの初期化（手順1）
 for(i = 0; i < N; i++) {
  node[i].d = Inf;
  node[i].c = ?1;
 }
 node[start].d = ?2;
 node[start].c = ?3;
 node[start].h = -1;
```

```
while (1) {
  // 解候補から距離値最小のものs（解）を見つける（手順2.1）
  s = -1;
  dmin = Inf;
  for (i = 0; i < N; i++) {
   if (node[i].c == Gray && node[i].d < dmin){
    dmin = ?4;
    s = ?5;}}

  // 見つからなければ終了（手順2.2）
  if (s == -1) break;
  // 手順2.3
  node[s].c = ?6; // sを解とする
```

アルゴリズムの実装

```
    for (i = 0; i < N; i++)   // 解となっていない、sに隣接する頂点iを解候補とし距離を更新
      if (node[i].c != ?7 && a[s][i] < ?8) {
        if (node[i].d > node[s].d + a[s][i]){
          node[i].d = node[s].d + a[s][i];
          node[i].h = s;
          node[i].c = Gray;}
      }
  }
}
```

manaba小テスト：06-2

- 15分
- 8点

プログラム

```
void PrintResult(int start)
{
  printf("%dから各頂点までの最短距離（経路）¥n", start);
  for(int no_end = 0; no_end < N; no_end++){
    printf("%3d ¥(", node[no_end].d);
    for(int i = no_end; ; i = node[i].h){
      printf("%d", i);
      if(node[i].h == -1) break;
      printf("<-");}
    printf(")¥n");}
}
```

プログラム

```
#define Start_No 0

int main(void)
{
  Dijkstra(Start_No);
  PrintResult(Start_No);
  return 0;
}
```

実行例

```
./a.out
0から各頂点までの最短距離(経路)
 0 (0)
19 (1<-2<-0)
 7 (2<-0)
17 (3<-2<-0)
22 (4<-3<-2<-0)
28 (5<-6<-3<-2<-0)
26 (6<-3<-2<-0)
```

```
./a.out
1から各頂点までの最短距離(経路)
19 (0<-2<-1)
 0 (1)
12 (2<-1)
 7 (3<-1)
12 (4<-3<-1)
18 (5<-6<-3<-1)
16 (6<-3<-1)
```