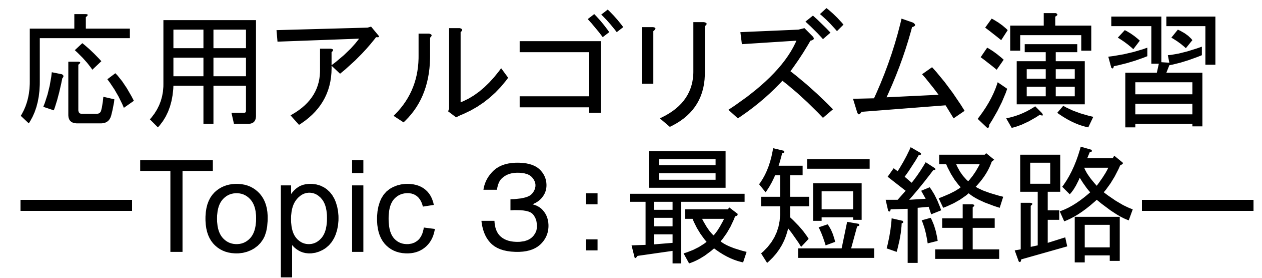
応用アルゴリズム演習
―Topic 3：最短経路―



鎌田十三郎

本日の内容

- 基本データ型: 優先度キュー
  - 実装: heap
  - comparator
  - 参考: qsort の紹介
- グラフに対する最短経路問題
  - ダイクストラ (Dijkstra) 法
  - A* アルゴリズム

優先度キュー (priority queue)

- 優先度の高いものから取り出すことのできるキュー
  - void enqueue(q, elem): q に elemを追加
  - ELEM dequeue(q): q から最優先の要素を取り出し
- 実装: ヒープソートで用いるヒープを利用
  - 守るべきルール:「親」は「子」より前
  - enqueue: 最後に要素を加え upheap
  - dequeue: 先頭削除＆
最後要素を先頭に＆downheap

比較器 (Comparator)　1/2

- 提供した優先度キューは汎用
  - 要素型は ELEM, 適当に要素型を変えてください
  - 要素の大小比較は、自分で実装してね
    int compare(a, b):
    - a,b 順で OK なら負の数を
    - a, b 大きさ同じなら 0
    - b, a の順に変えるべきなら正の数
    - 但し、a, b は、実際には要素へのポインタ値

```
/* int を小さい順に並べる場合 */
int compare(int * a, int * b) {
    if(*a < *b) return -1;
    if(*a > *b) return 1;
    return 0;
}
```

比較器 (Comparator)　2/2

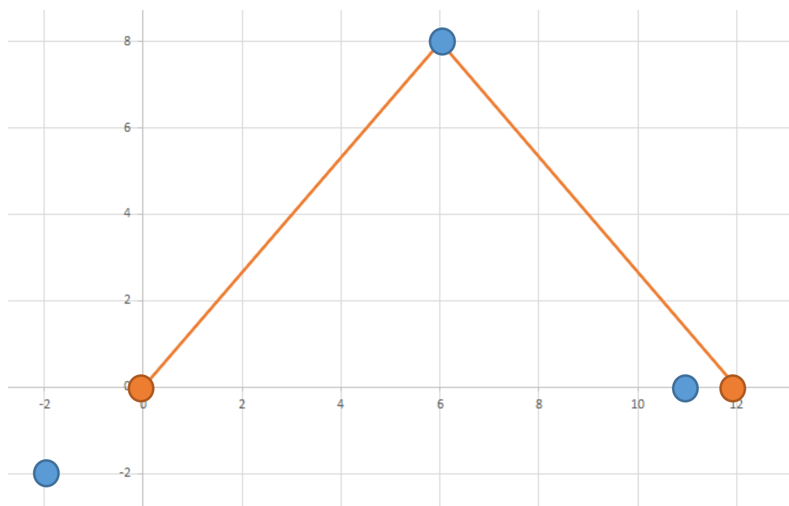
比較器を用いたライブラリもある。

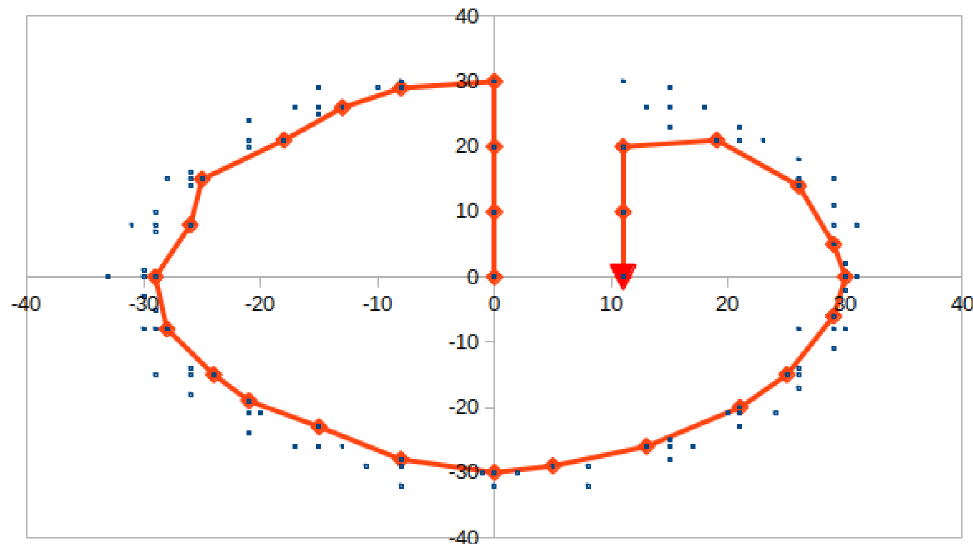
- 標準ライブラリ: qsort, 配列のソートをしてくれる
  - void qsort(void * base, size_t num, size_t size,
                int (*compar)(const void*, const void*));
    - 汎用ライブラリであるため、対象データに合わせて、サイズや型変換が必要。（コード例は web にて）
  - base: 配列の先頭アドレス
  - num: 要素数
  - size: 各要素の大きさ(sizeof)
  - compar: ２要素へのポインタをもらい、比較結果を返す関数（へのポインタ）

最短経路問題：例題

- 盤面上にいくつか点がある
  - ●２点の距離が10以下の場合は移動可能
  - ●始点から終点まで、最短経路で移動したい

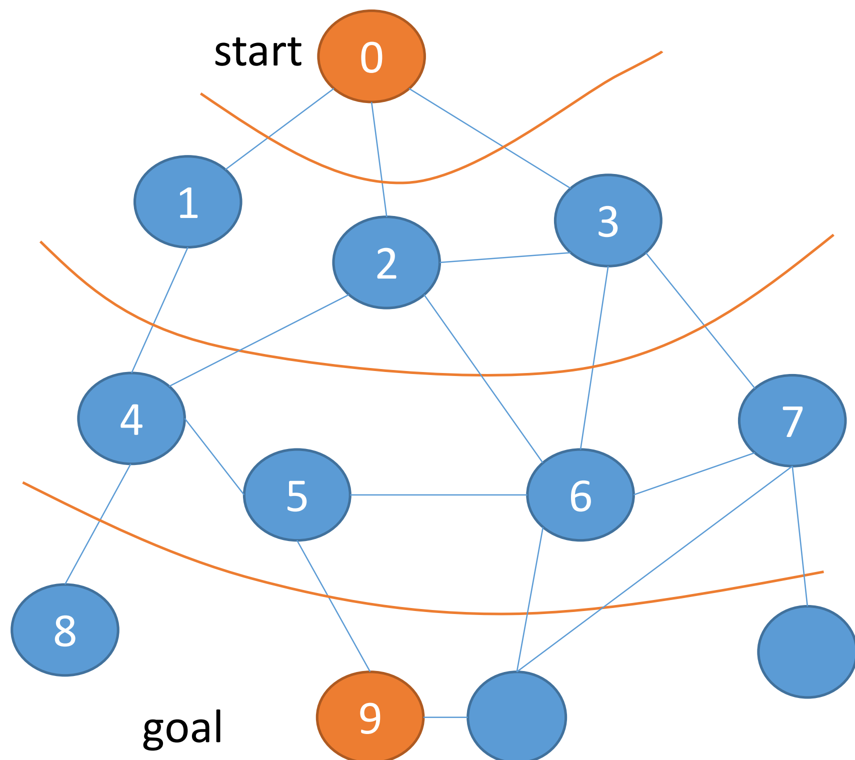
最短経路を
求めましょう





前回のお題(幅優先探索)

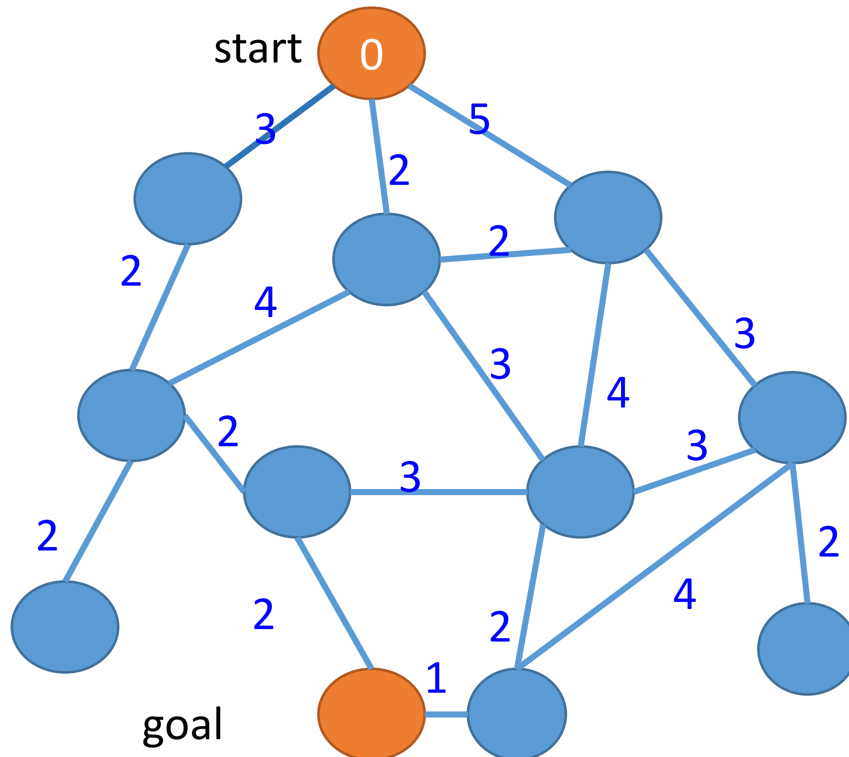
- 幅優先探索で問題を解いてみよう
  - start から goal まで、何ステップでいけるか考えよう!




```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        未訪問の隣接node があれば、
        enqueue(queue, node);
    }
}
```

もし、枝に長さがあったら？

- 一番近いところを探すのがちょっと面倒？
- でも、近いところからやらないと、何度もやり直し

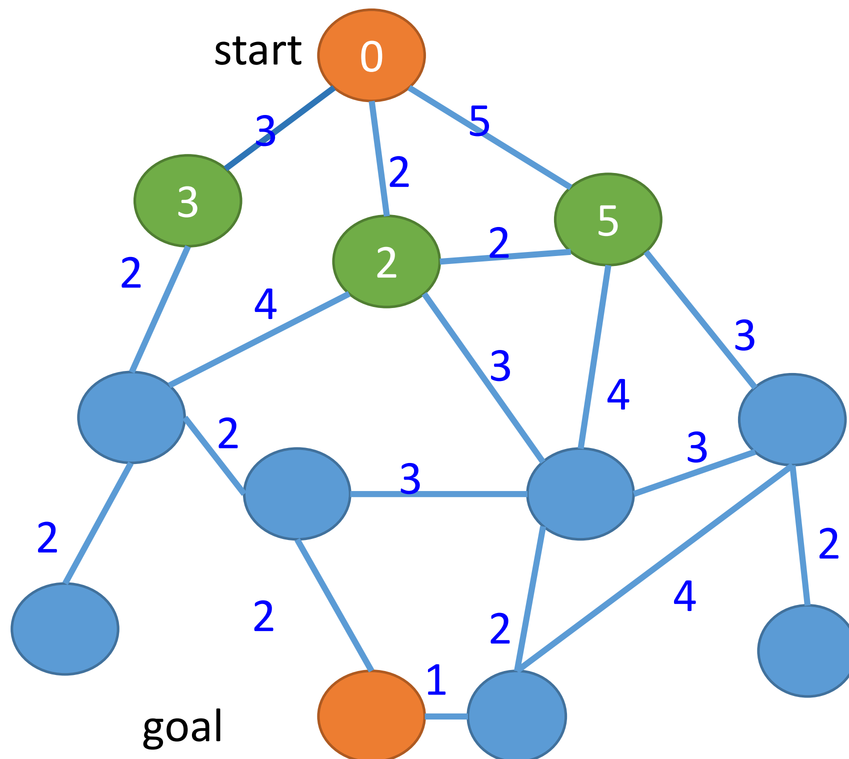



```
int solve() {
  enqueue(queue, 0 );
  while(qSize(queue)>0) {
    node_t here = dequeue(queue);
    if(ゴール?) return 答え;
    隣接node への経路候補が
    あれば、enqueue();
  }
}
```

ダイクストラ法 (Dijkstra's algorithm)

- start から近いところから、確定させていこう
  - queue から、一番近いものが取り出せれば

優先度キューでOK

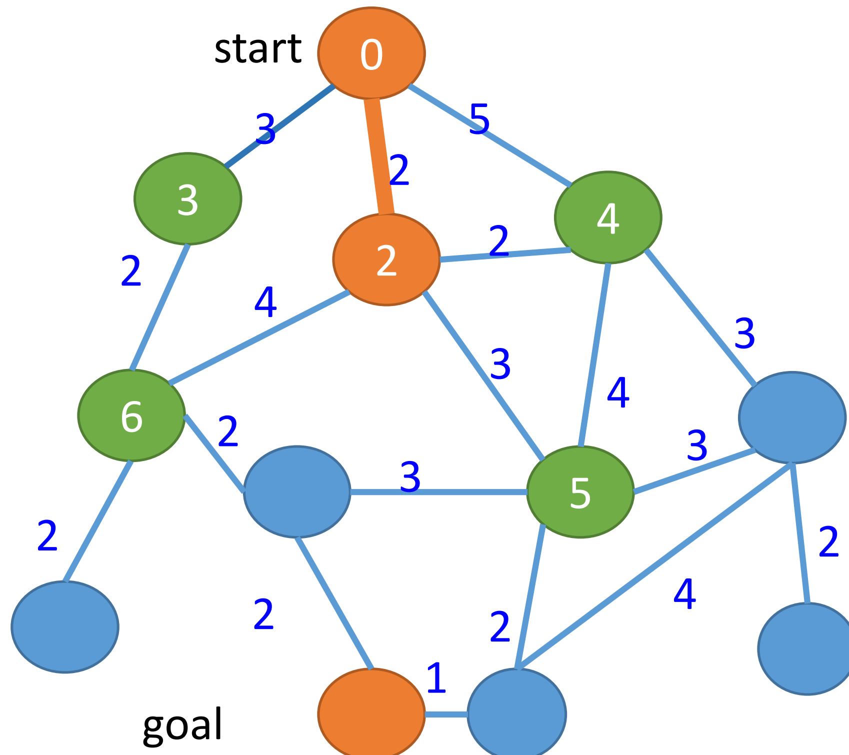



```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        隣接node への経路候補が
        あれば、enqueue();
    }
}
```

ダイクストラ法 (Dijkstra's algorithm)

- start から近いところから、確定させていこう
  - queue から、一番近いものが取り出せれば

優先度キュー でOK

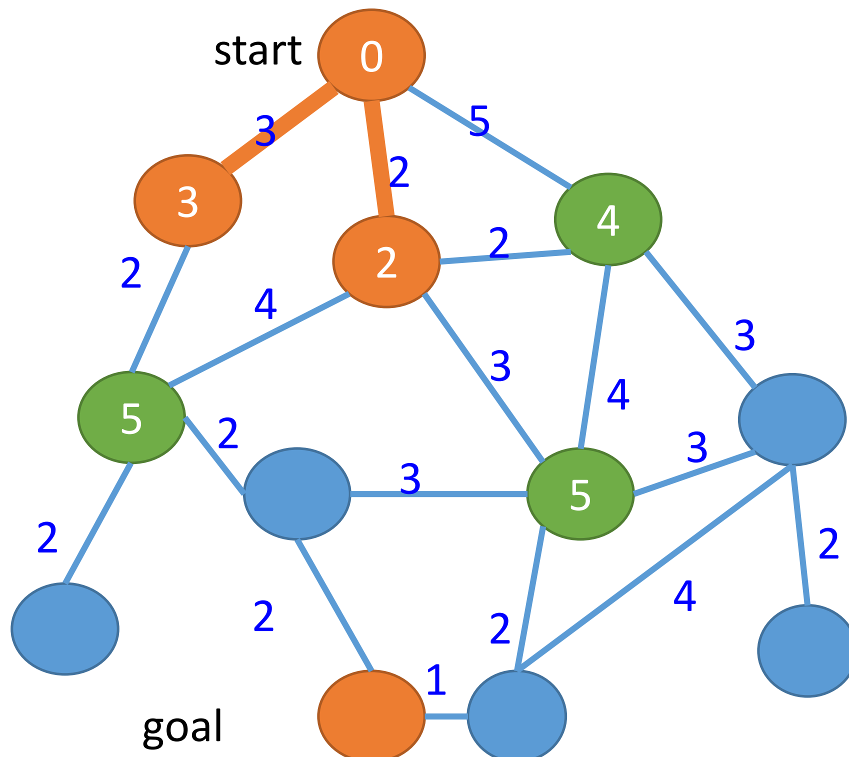



```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        隣接node への経路候補が
        あれば、enqueue();
    }
}
```

ダイクストラ法 (Dijkstra's algorithm)

- start から近いところから、確定させていこう
  - queue から、一番近いものが取り出せれば

優先度キュー でOK

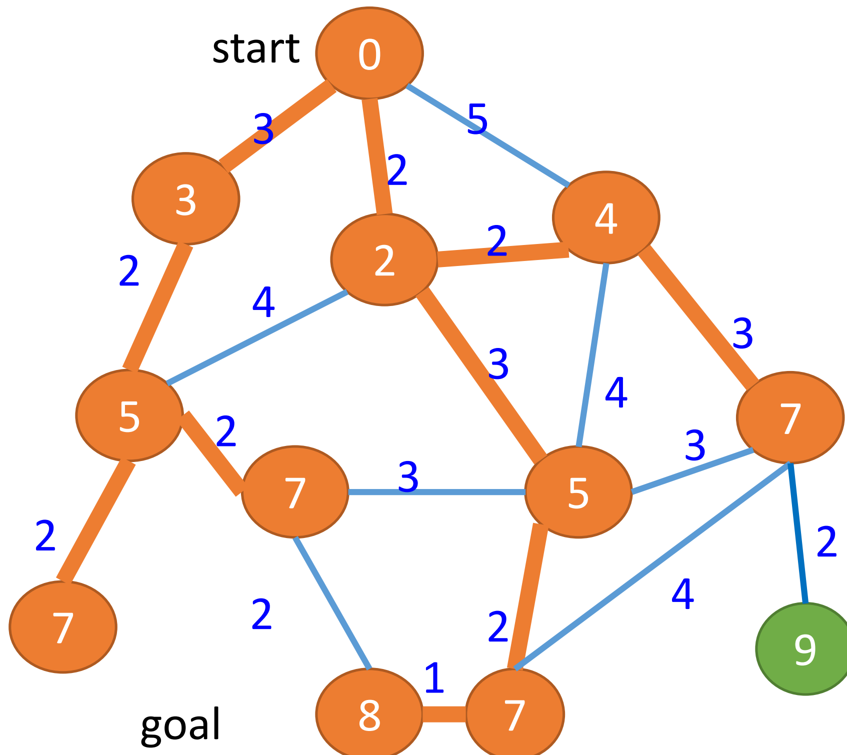



```
int solve() {
   enqueue(queue, 0 );
   while(qSize(queue)>0) {
       node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        隣接node への経路候補が
       あれば、enqueue();
    }
}
```

ダイクストラ法 (Dijkstra's algorithm)

- start から近いところから、確定させていこう
  - queue から、一番近いものが取り出せれば

優先度キュー でOK

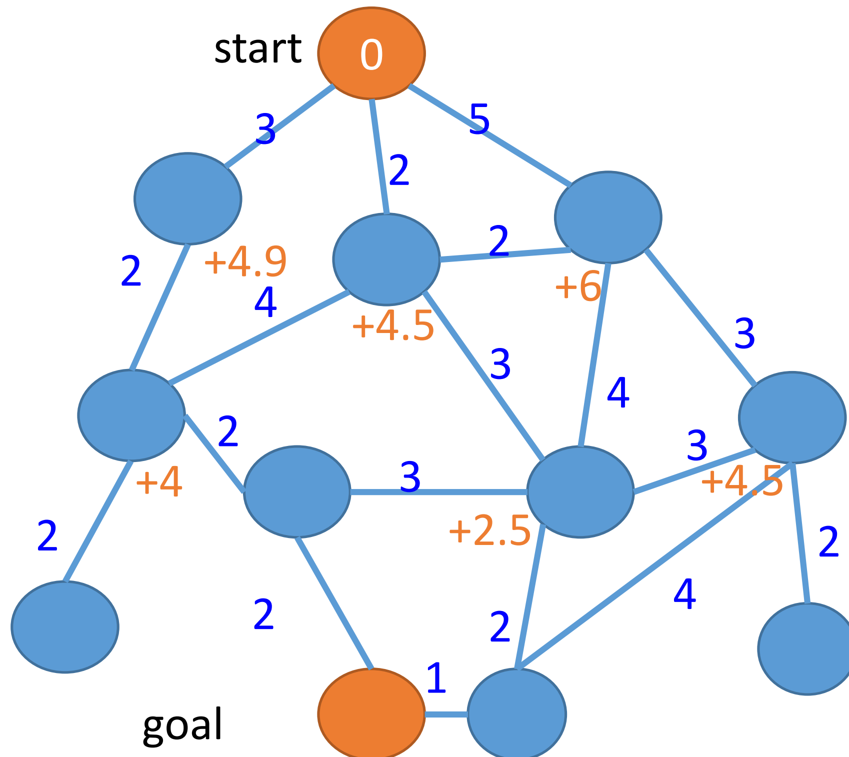



```
int solve() {
   enqueue(queue, 0 );
   while(qSize(queue)>0) {
      node_t here = dequeue(queue);
      if(ゴール?) return 答え;
      隣接node への経路候補が
      あれば、enqueue();
   }
}
```

A* (A-star)アルゴリズム

- goal への「見積もり値」を加味して小さい順に
  - 例えば、「goal への直線距離」を利用

「各ノードまでの経路」＋「goal への見積もり値」で順位付け

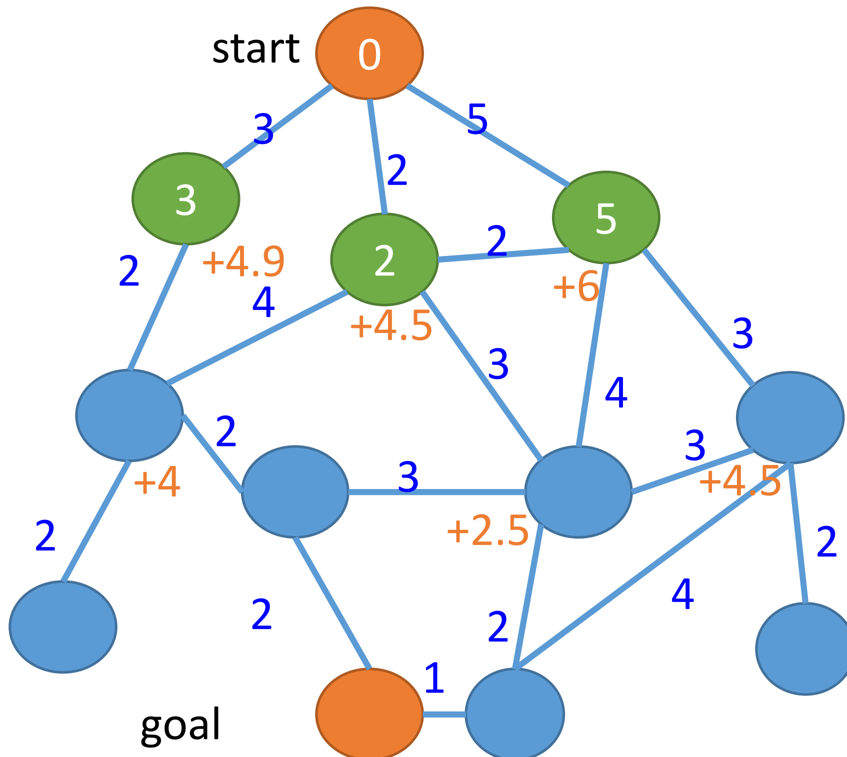



```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
         if(ゴール?) return 答え;
         隣接node への経路候補が
        あれば、enqueue();
    }
}
```

A* (A-star)アルゴリズム

- goal への「見積もり値」を加味して小さい順に
  - ●例えば、「goal への直線距離」を利用

「各ノードまでの経路」＋「goal への見積もり値」で順位付け

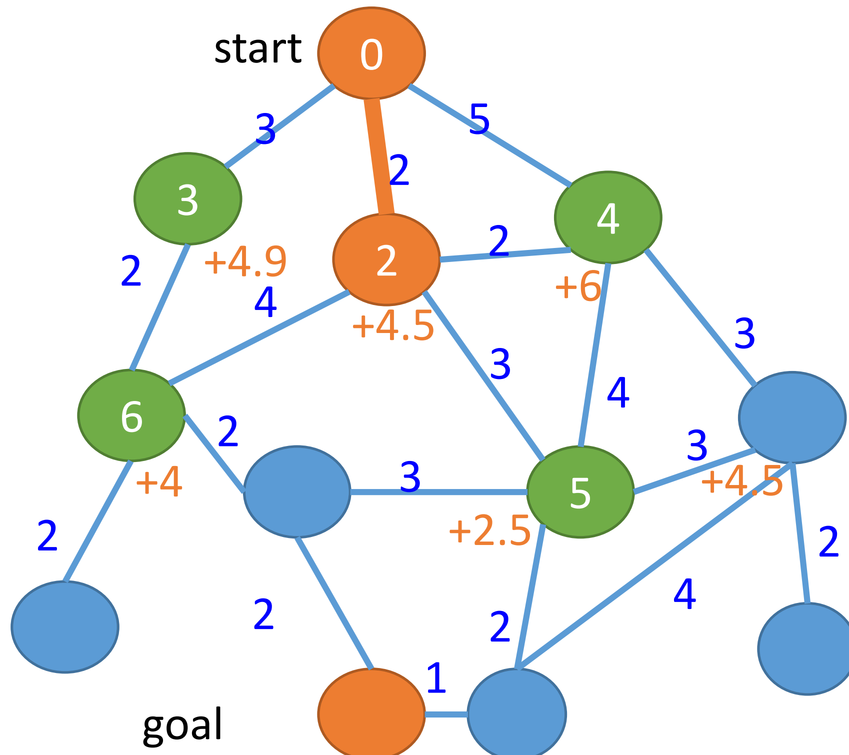



```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
         if(ゴール?) return 答え;
         隣接node への経路候補が
        あれば、enqueue();
    }
}
```

A* (A-star)アルゴリズム

- goal への「見積もり値」を加味して小さい順に
  - ●例えば、「goal への直線距離」を利用

「各ノードまでの経路」+
「goal への見積もり値」
で順位付け

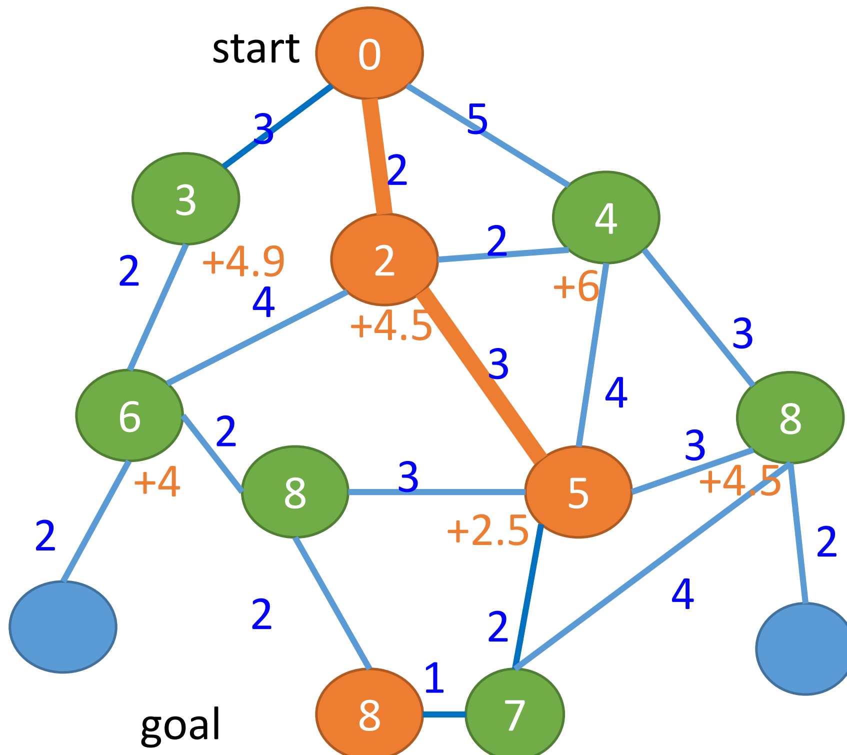



```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        隣接node への経路候補が
        あれば、enqueue();
    }
}
```

A* (A-star)アルゴリズム

- goal への「見積もり値」を加味して小さい順に
  - ●例えば、「goal への直線距離」を利用

「各ノードまでの経路」＋
「goal への見積もり値」
で順位付け

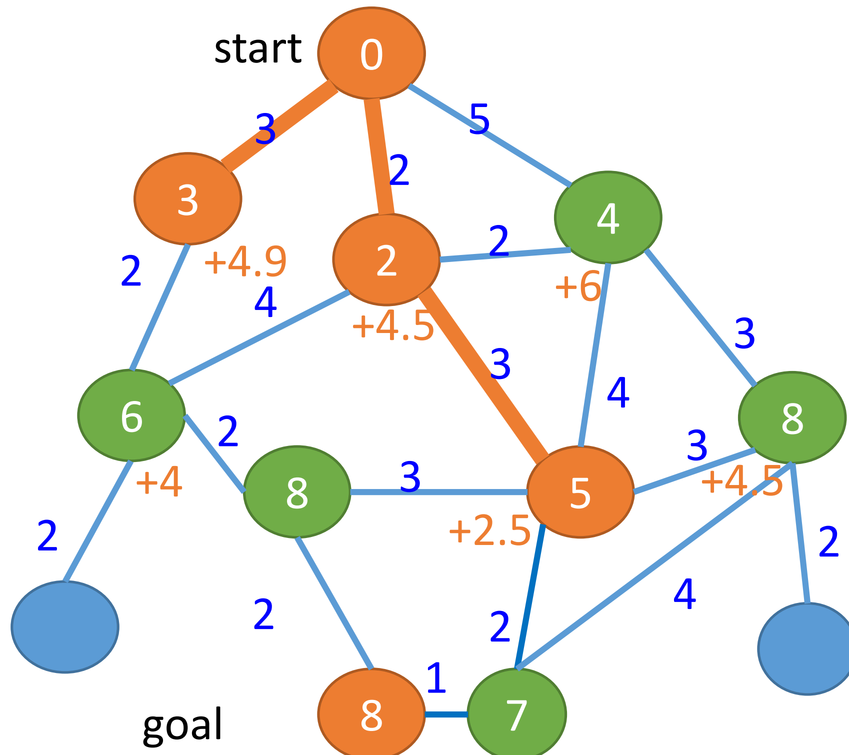



```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
         if(ゴール?) return 答え;
         隣接node への経路候補が
        あれば、enqueue();
    }
}
```

A* (A-star)アルゴリズム

- goal への「見積もり値」を加味して小さい順に
  - 例えば、「goal への直線距離」を利用

「各ノードまでの経路」＋「goal への見積もり値」で順位付け

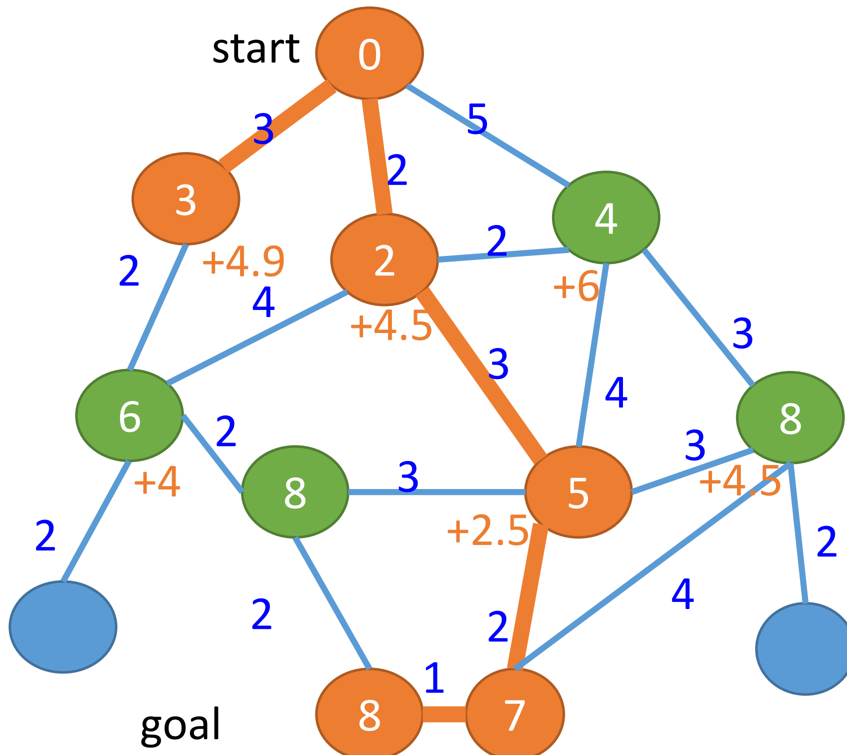



```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        隣接node への経路候補が
        あれば、enqueue();
    }
}
```

A* (A-star)アルゴリズム

- goal への「見積もり値」を加味して小さい順に
  - 例えば、「goal への直線距離」を利用

「各ノードまでの経路」＋「goal への見積もり値」で順位付け




```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        隣接node への経路候補が
        あれば、enqueue();
    }
}
```