[プログラム例@github]

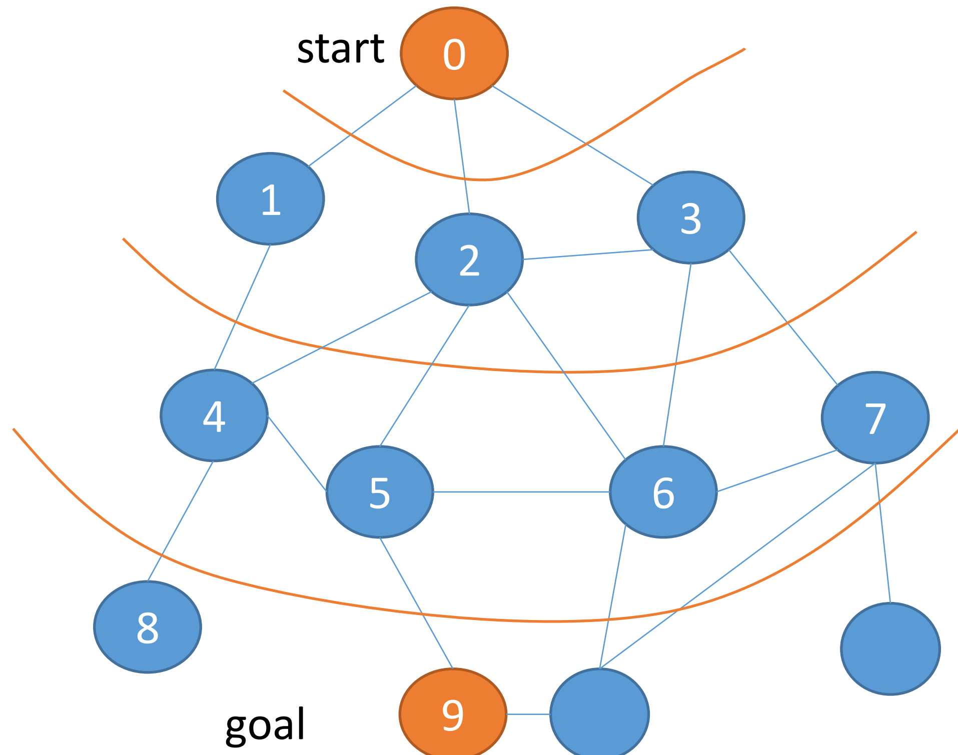
幅優先探索、最短経路探索

お題（迷路と命ず）

- 問題：ACM ICPC というプログラミングコンテストの2010年国内予選問題（参考 1, 2, AOJ）

幅優先探索という考え方

- 開始点から近い順に、隣接点を探索していく方法
  - 一度辿ったところは、再探索しない




```
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 に対して
            enqueue(queue, node);
    }
}
```

深さ優先探索と幅優先探索

■深さ優先探索

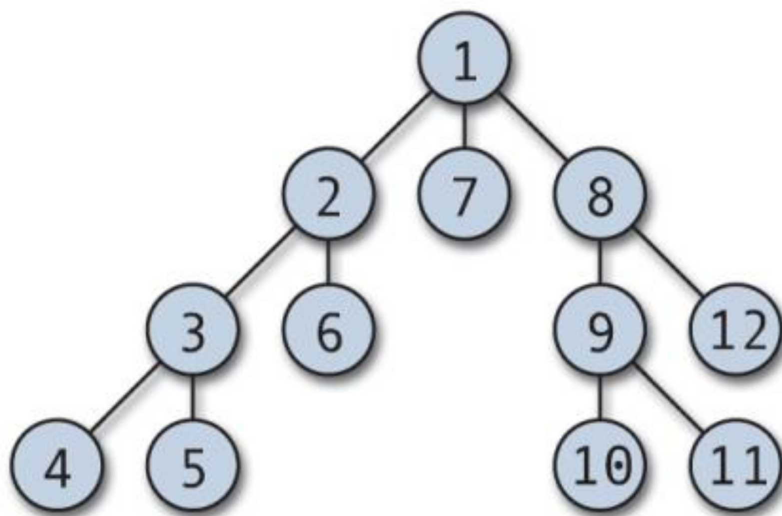
●最近探索した点を優先

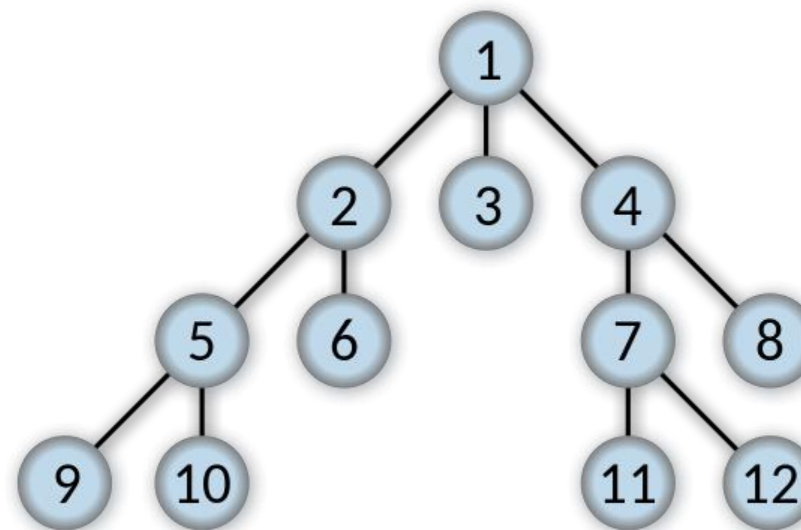
●Stack or 関数再帰呼出し利用

■幅優先探索

●開始点から近い順

●Queue を利用

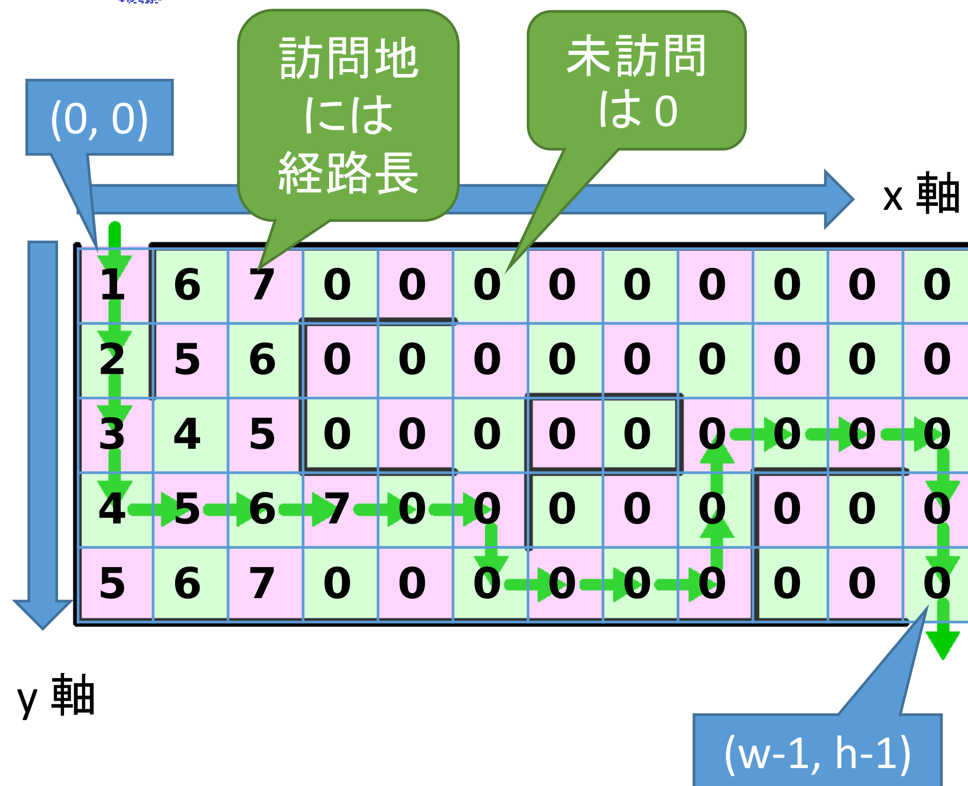




それぞれの図は Wikipedia から引用

幅優先探索してみる

- なにがいるかな？
  - 各方向に進めるか？
    - 壁判定がいりそう
  - 開始点から順番に探索
    - queue が便利
  - 各地点は訪問済み？ステップ数の記録は？
    - 2次元配列でOK



幅優先探索してみる

プログラム例@github

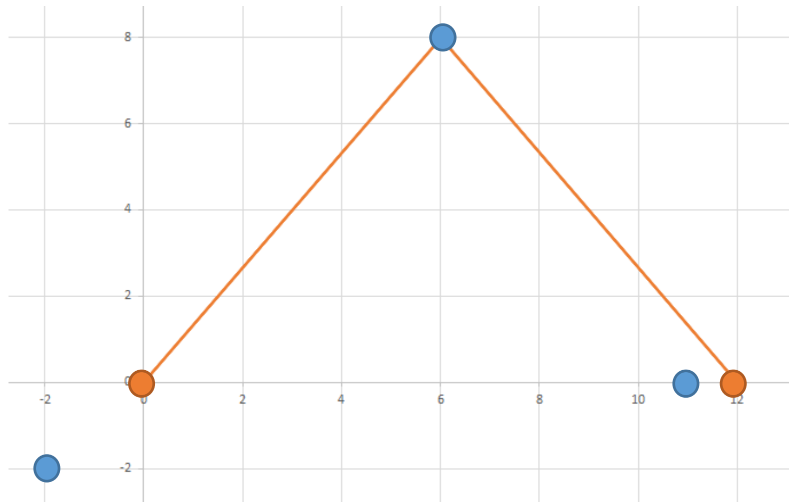
- なにがいるかな？
  - 各方向に進めるか？
    - 壁判定がいりそう
  - 開始点から順番に探索
    - queue が便利
  - 各地点は訪問済み？ステップ数の記録は？
    - 2次元配列でOK

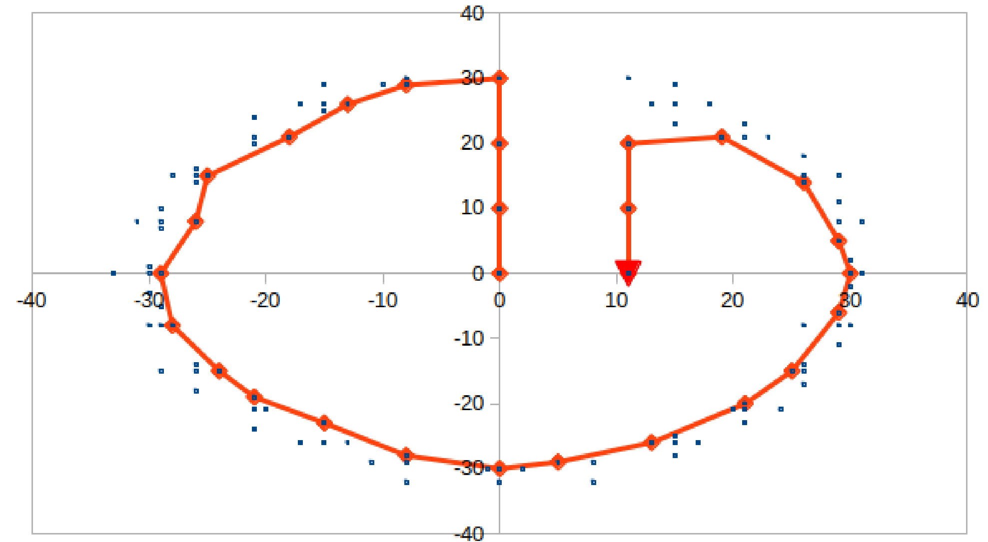
```
int solve(....) {
    q.push(start);
    while( !q.empty() ) {
        point here = q.front();
        q.pop();
        for( d: directions) {
            if(d の方向にいけたら) {
                point next = here + d;
                next がゴールなら終了。
                if(next は最初の訪問) {
                    next の訪問登録;
                    q.push(next);
                }
            }
    }
    // 辿りつかなかった場合の処理
}
```

問題2つめ（最短経路問題）

- 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

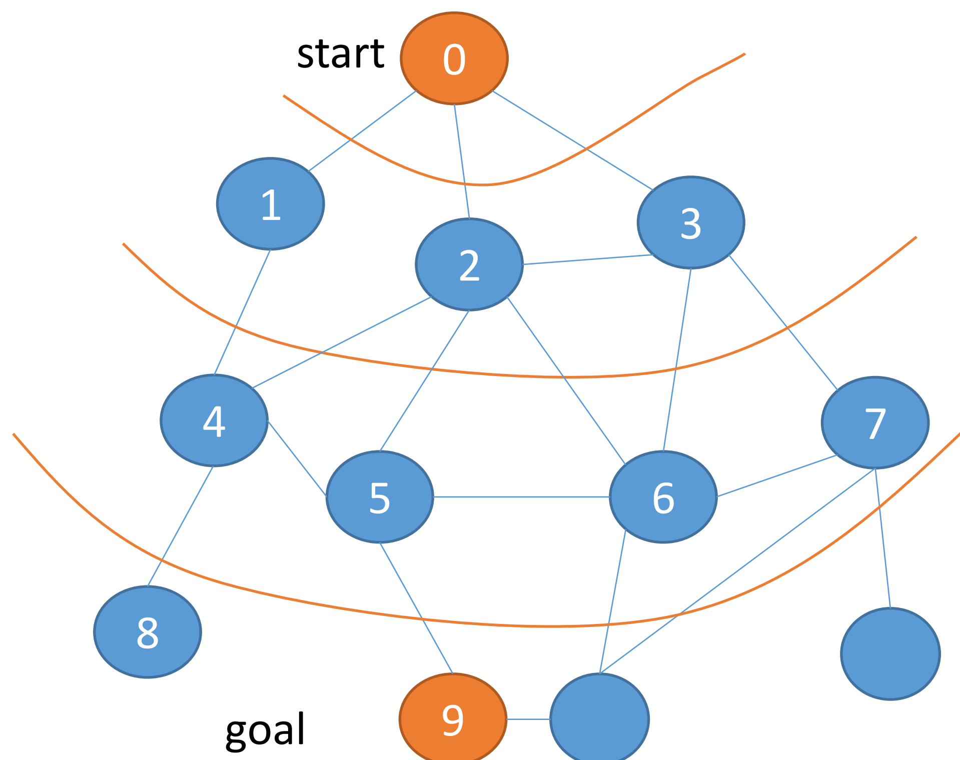




幅優先探索という考え方

再掲

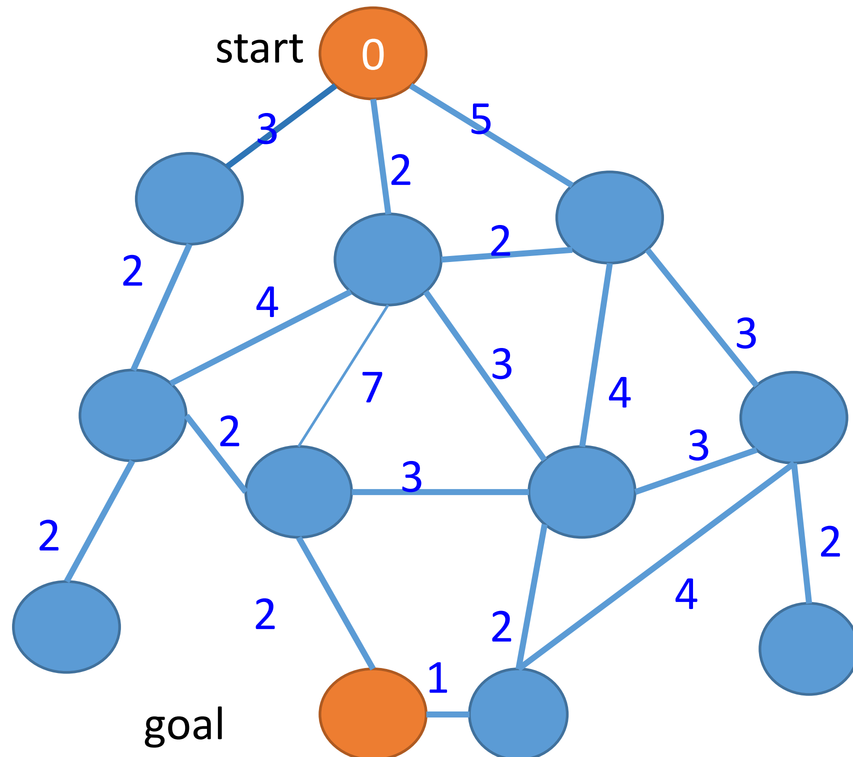
- 開始点から近い順に、隣接点を探索していく方法
  - 一度辿ったところは、再探索しない




```
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 に対して
            enqueue(queue, node);
    }
}
```

今回は枝に長さがある

- 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



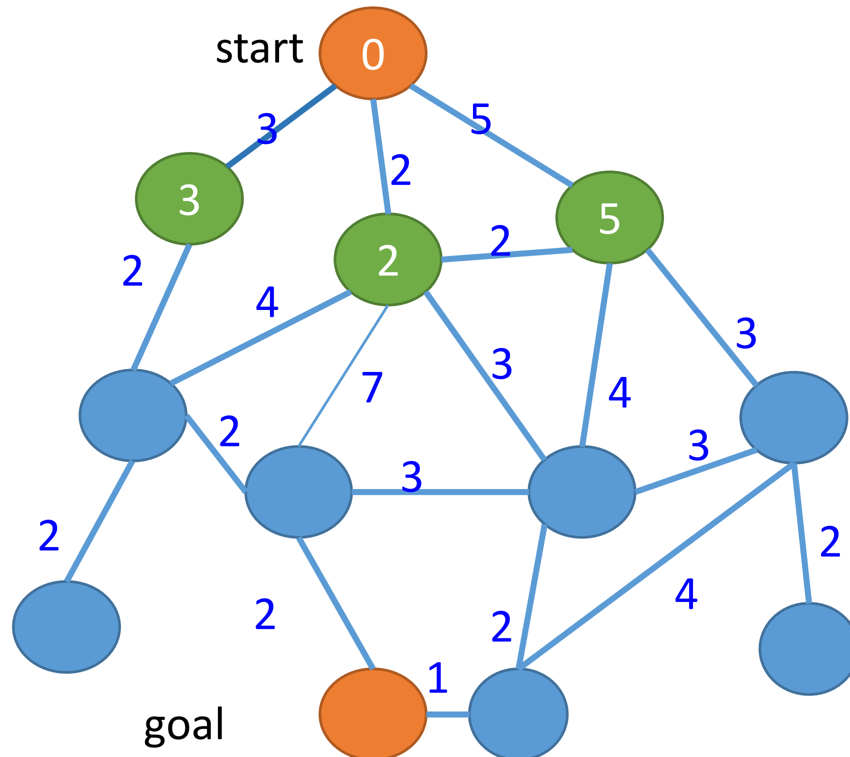

```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        未訪問の隣接node に対して
            enqueue();
    }
}
```

ダイクストラ法 (Dijkstra's algorithm)

プログラム例@github

- 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

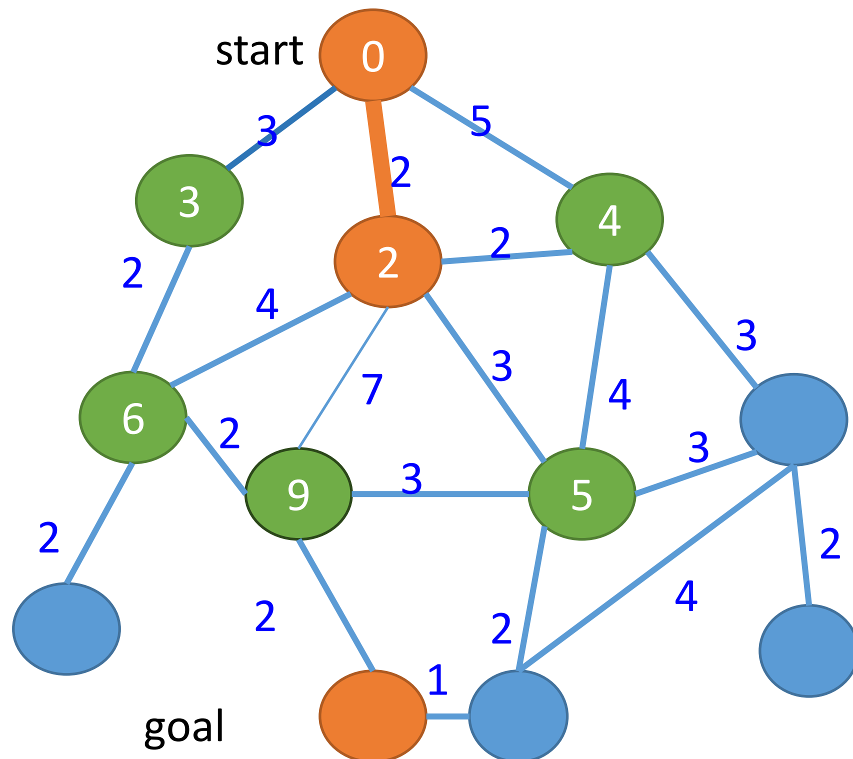



```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        未訪問の隣接node に対して
            enqueue();
    }
}
```

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

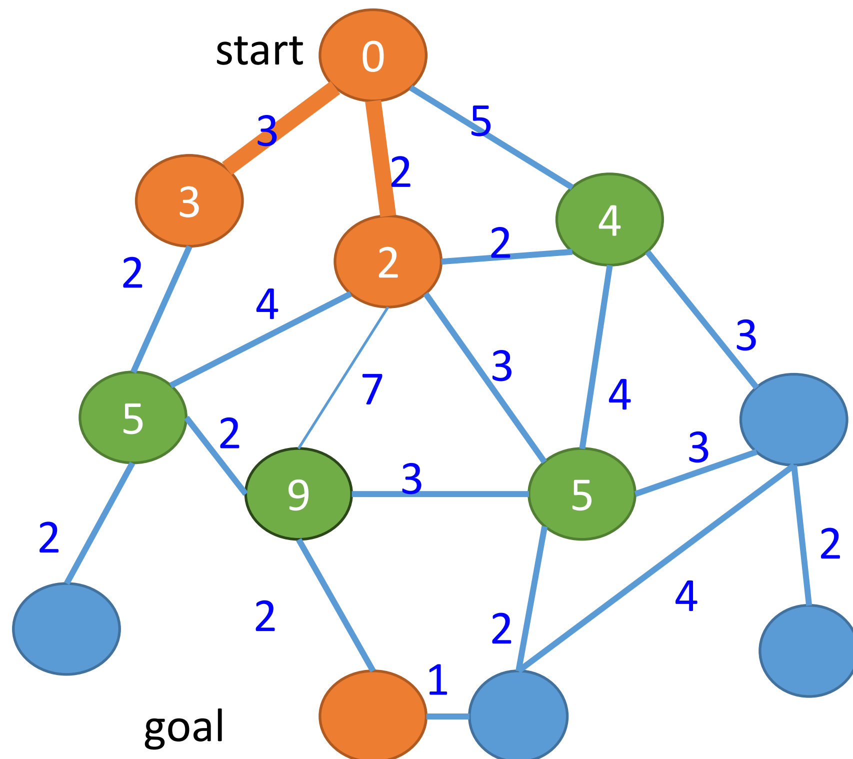



```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
         if(ゴール?) return 答え;
         未訪問の隣接node に対して
             enqueue();
    }
}
```

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

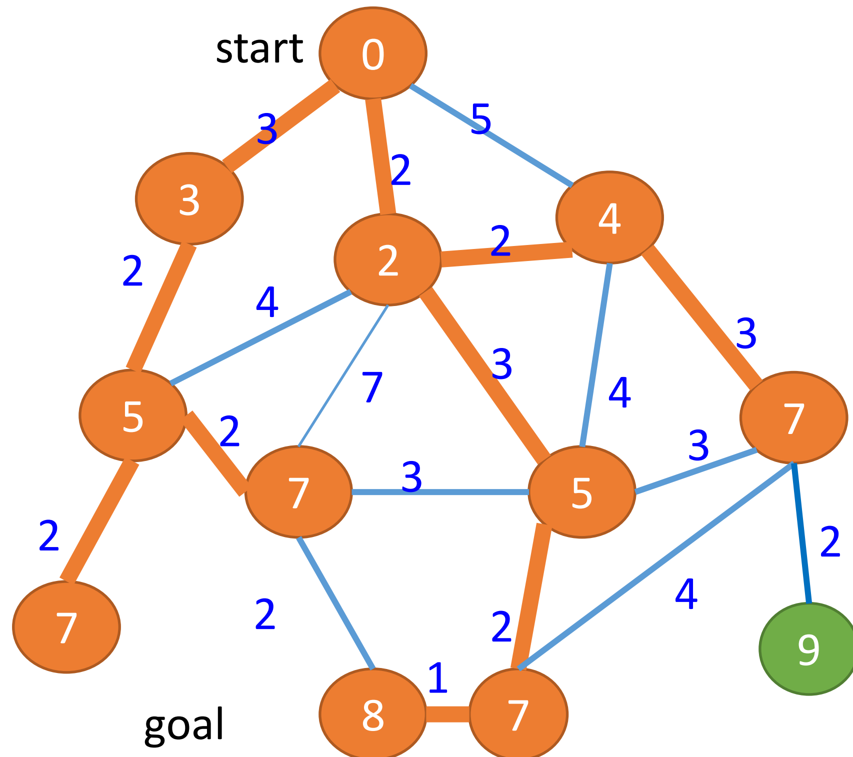



```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
         if(ゴール?) return 答え;
         未訪問の隣接node に対して
             enqueue();
    }
}
```

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

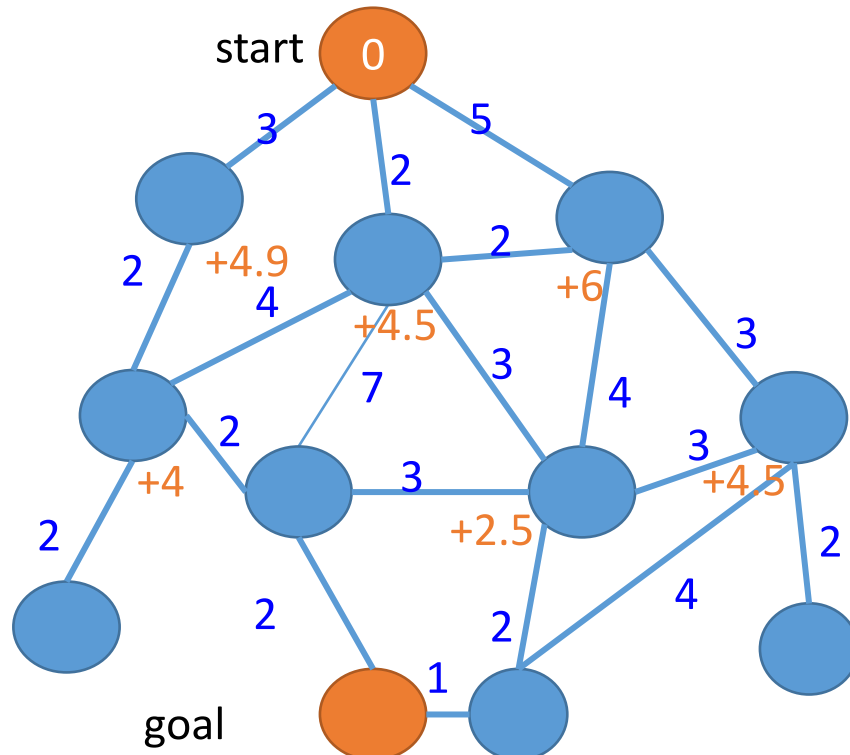



```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        未訪問の隣接node に対して
            enqueue();
    }
}
```

A* (A-star)アルゴリズム

プログラム例@github

- 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

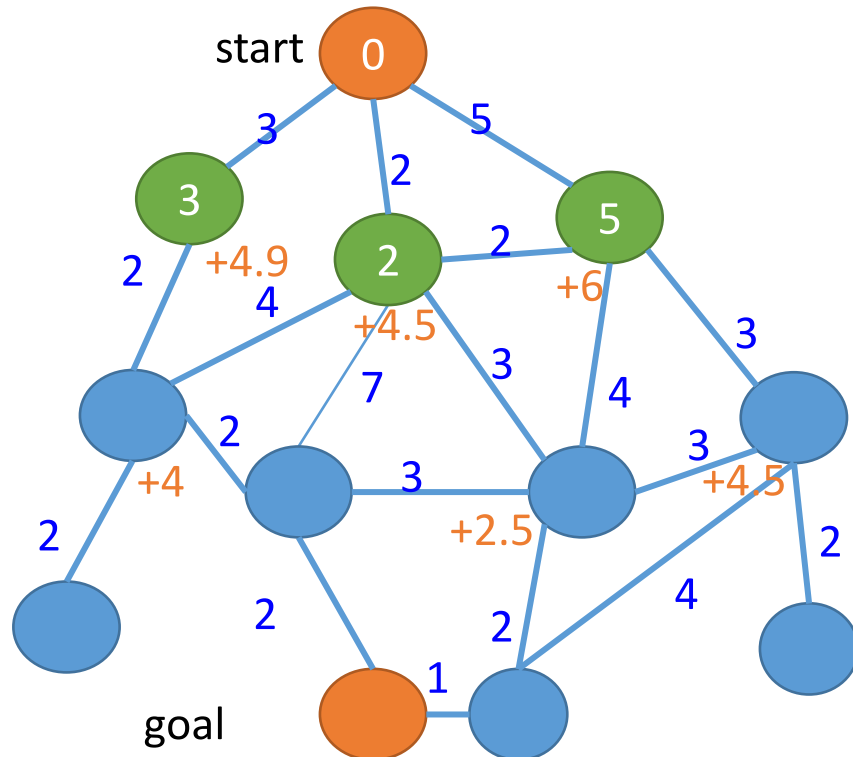


「各ノードまでの経路」＋「goal への見積もり値」で順位付け

```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        未訪問の隣接node に対して
            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

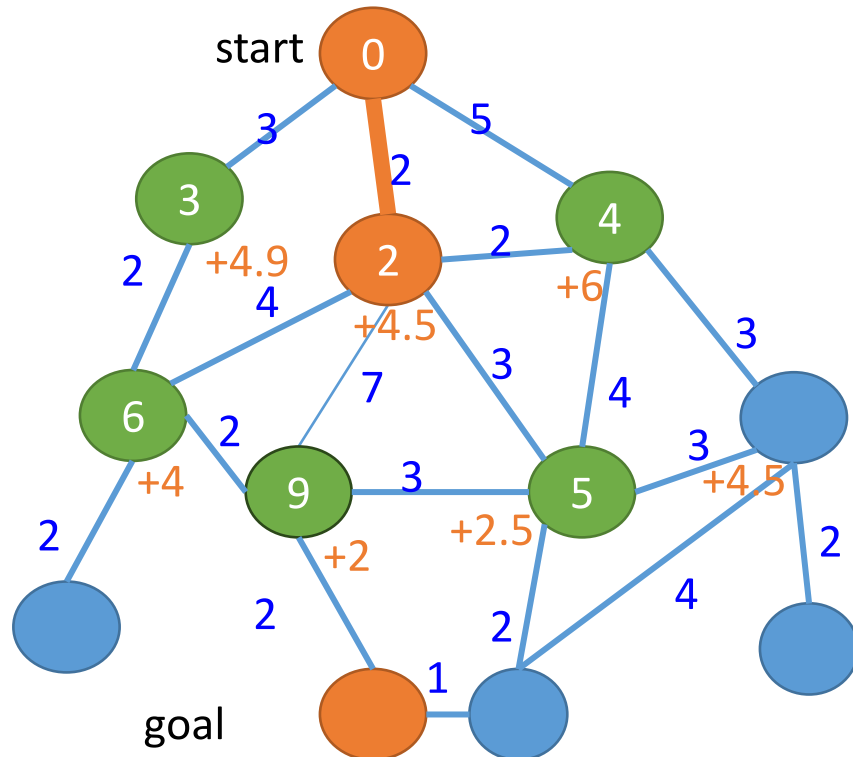


「各ノードまでの経路」＋「goal への見積もり値」で順位付け

```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        未訪問の隣接node に対して
            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

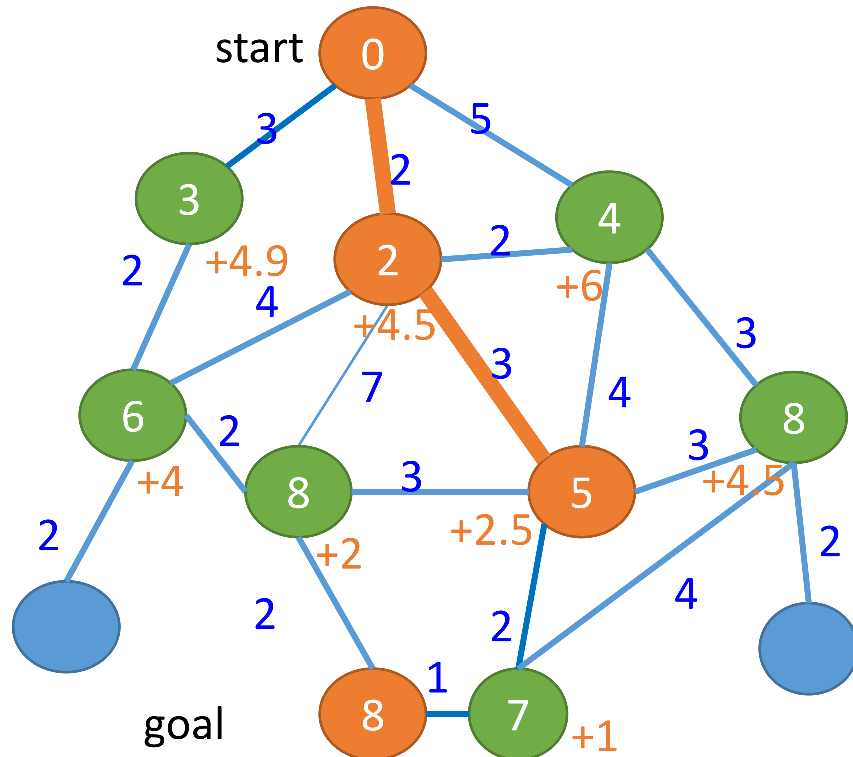


「各ノードまでの経路」＋「goal への見積もり値」で順位付け

```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        未訪問の隣接node に対して
            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



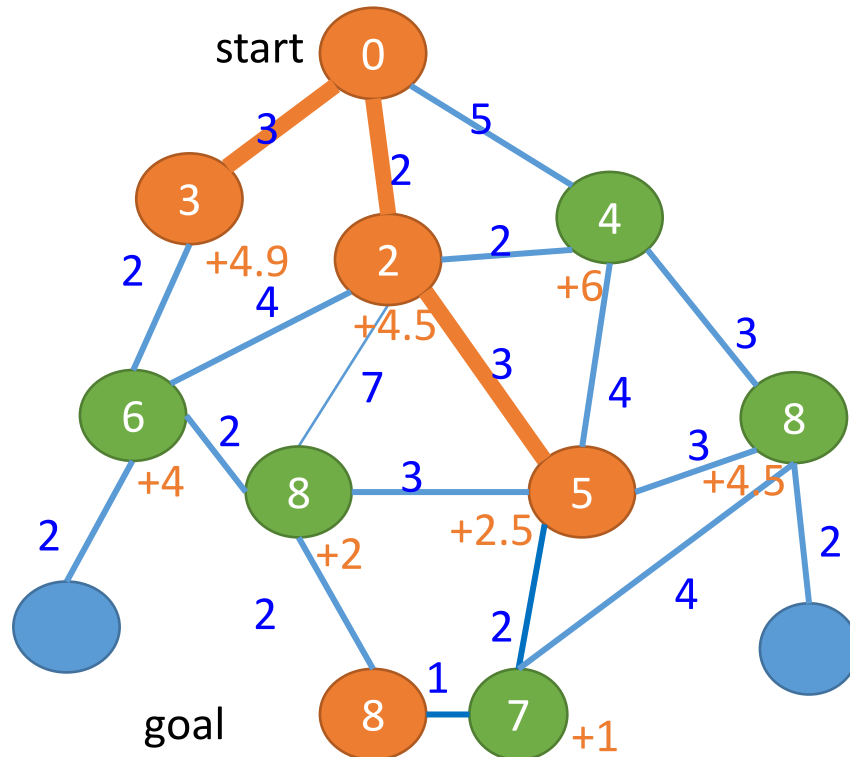
「各ノードまでの経路」＋
「goal への見積もり値」
で順位付け

```
int solve() {
    enqueue(queue, 0 );
    while(qSize(queue)>0) {
        node_t here = dequeue(queue);
        if(ゴール?) return 答え;
        未訪問の隣接node に対して
            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

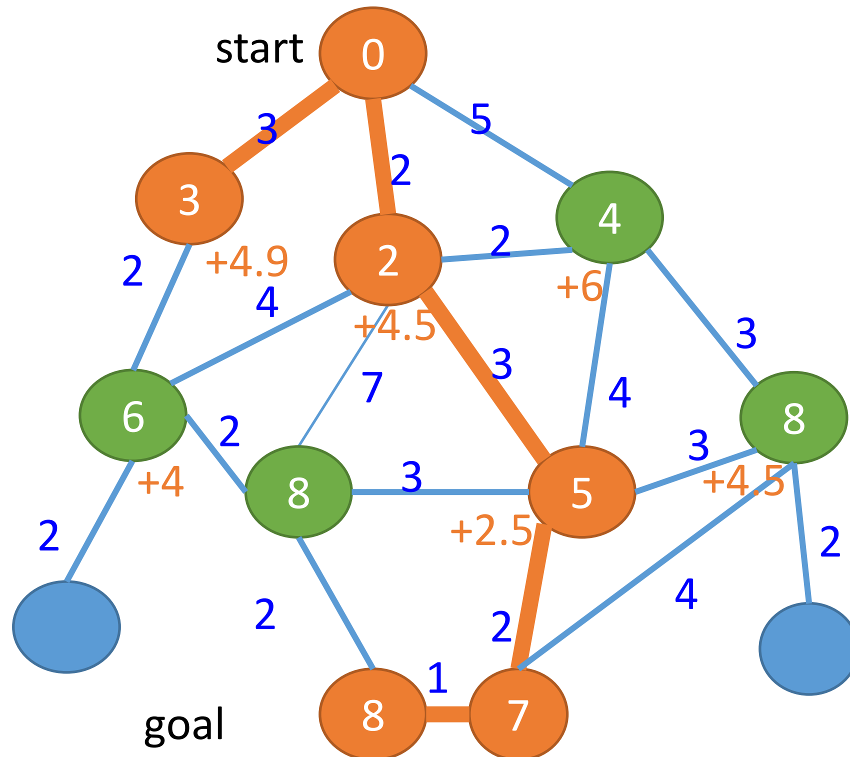
「各ノードまでの経路」＋
「goal への見積もり値」
で順位付け




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

ACM ICPC 2007 国内予選

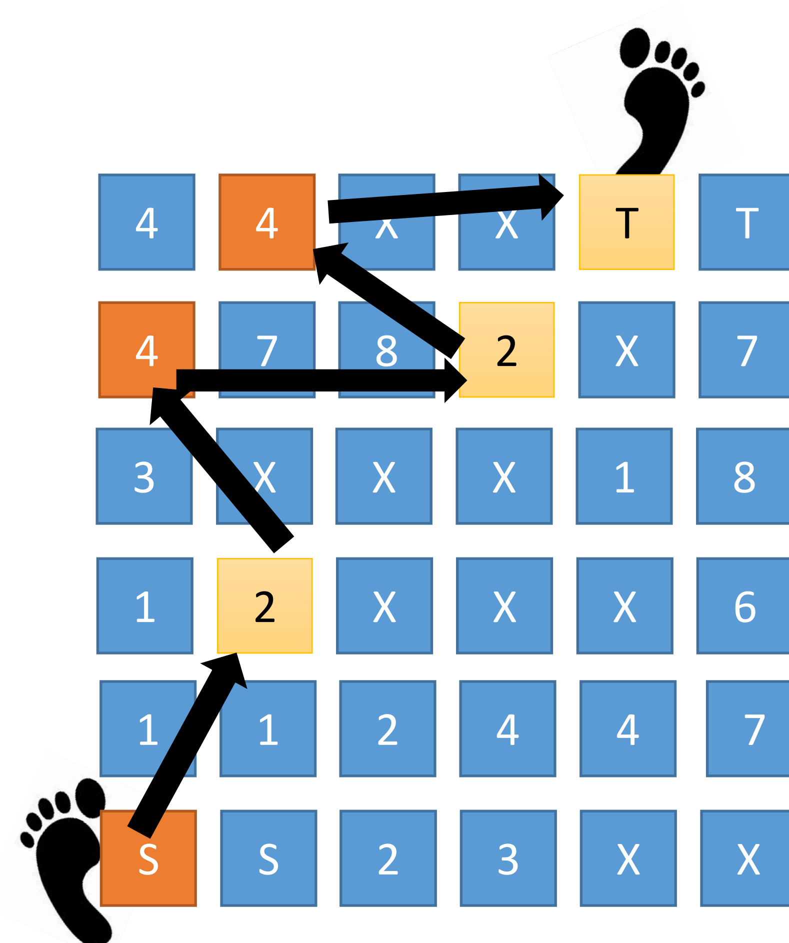
さまざまな最短経路問題

- スタートエリアからゴールエリアに移動
  - 各位置には、左足か右足を置ける
  - 各位置に書かれた分の時間を消費
    - 足を置けない場所もある
  - 次に置くのは逆の足
    - 置ける範囲が決まっている

4	4	X	X	T	T
4	7	8	2	X	7
3	X	X	X	1	8
1	2	X	X	X	6
1	1	2	4	4	7
S	S	2	3	X	X

解き方を考えてみよう

- スタートエリアからゴールエリアに移動
  - 各位置に、左足か右足を置く
  - 各位置にかかれた分の時間を消費
    - 足を置けない場所もある
  - 次に置くのは逆の足
    - 置ける範囲が決まっている
- 考え方？
  - 最短経路っぽい？
  - でも、左足右足って？
  - 同じ場所に両足置くことも？



ACM ICPC 2007 国内予選

考え方

プログラム例(C++)
プログラム例(python)

- [足の位置、左／右]をノードとするグラフをイメージすれば？
  - ノードをイメージできれば、最短経路問題