コンピュータ実習

第10回 2024年12月5日

カーナビゲーション4
最短経路探索

担当:相澤 直人, 宍戸 博紀

コンピュータ実習（第10回）

（復習）グラフィックスによる地図表示

アニメーションによる可視化

カーナビの画面

方角

現在地

地図の拡大・縮小

目的地までの距離

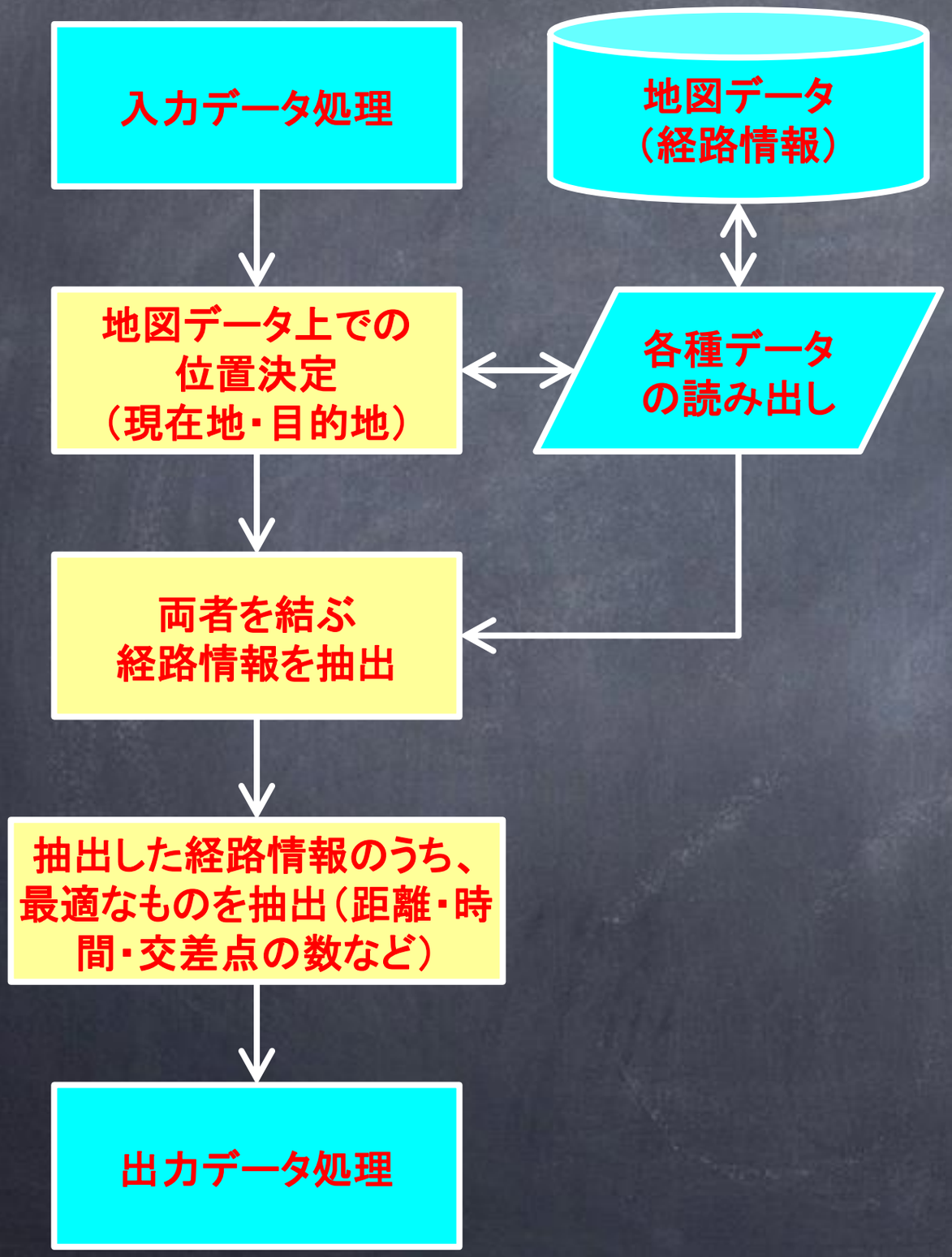
縮尺

現在地名

経路

コンピュータ実習（第10回）

カーナビ作成に向けて



地図データの検索と抽出
→ 入出力・変数・構造体

抽出したデータを元に経路を割り出す
→ 条件検索（サーチ）

割り出した経路のうち、最適なものを選ぶ
→ 条件並び替え（ソート）

コンピュータ実習（第10回）

今後の授業の進行

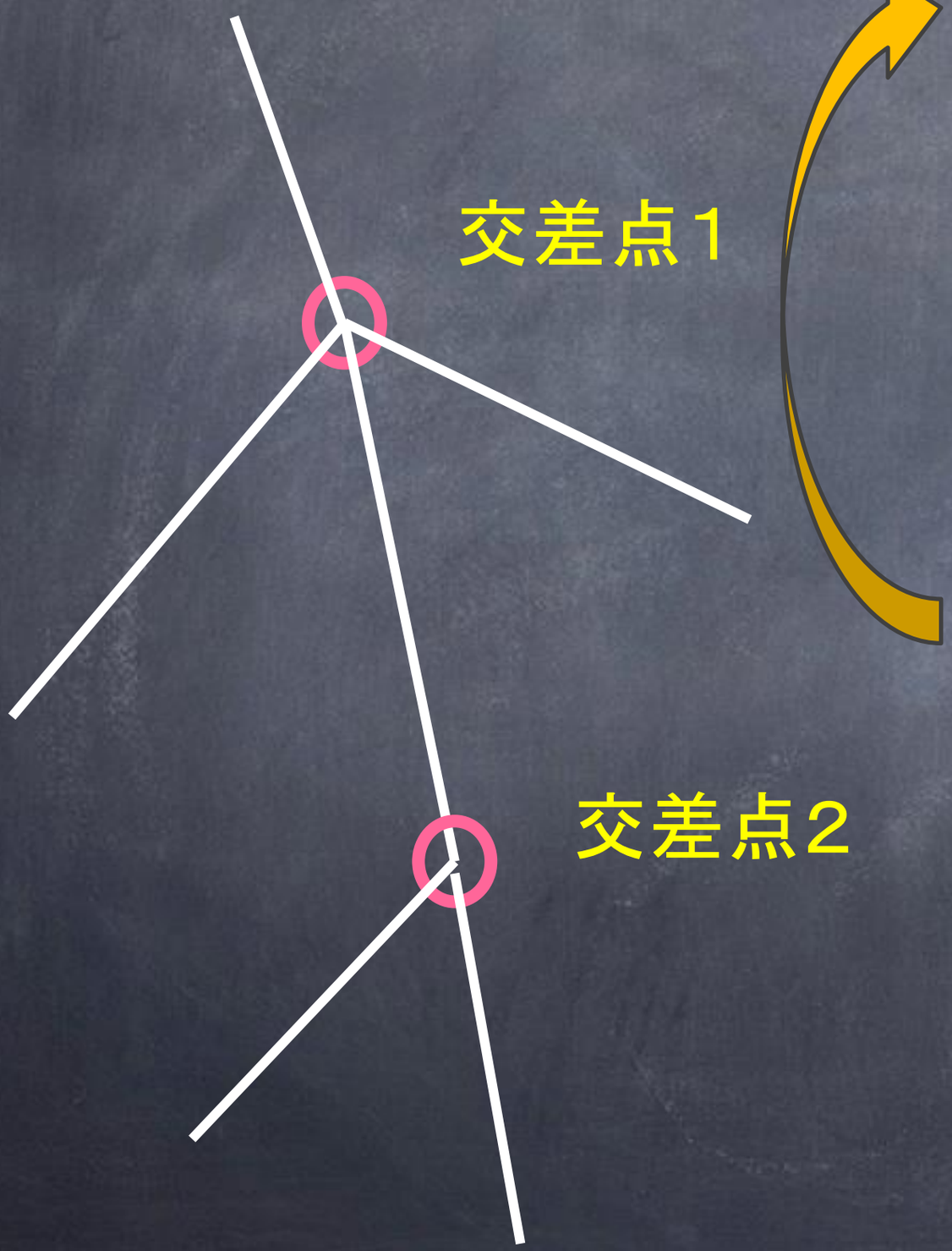
1. 地図データを元に、目的地を決定
   - 地図データの読み込み（11/14）
   - 交差点（地図データ）の検索（11/21）
2. 経路を探索
   - 条件に基づく経路の設定（11/21: 簡易版）
   - 最適経路の計算（距離・時間等）

本日の講義

3. 経路をドライバーに分かりやすく提示
   - 画面上で車をアニメーション表示（11/28, 12/12）

コンピュータ実習（第9回おさらい）

map.cのイメージ



1. 交差点位置に丸を書く
（円を書く関数を使う
関数の中身は第7回参照）
2. 交差点の名前を書く
(サンプルにある
draw_outtextxy関数を使う)
3. 隣接交差点の数だけ、隣接交差点の座標位置まで直線を引く
（for文を使う/OpenGLの機能で直線を引く）

コンピュータ実習（第9回おさらい）

関数map_show()について

```
/* 道路網の表示（新しく作成する部分） */
void map_show(int crossing_number) {
  /* 指定した交差点の数だけ、ウィンドウに交差点（地図）を表示する。
     まず各交差点には赤丸を描き、隣接交差点への道路を直線で描く */
  int i, j;
  /* 交差点ごとのループ*/
  for（i = 0; i < crossing_number; i++）{
    double x0 = cross[i].pos.x;
    double y0 = cross[i].pos.y;
  /* 交差点を表す円を描く */
  /* 交差点の名前を書く */
  /* 交差点から伸びる道路を描く */
  }
}
```

コンピュータ実習（第10回）

演習1　アニメーションによる可視化

交差点トレースプログラム

交差点を順にマーカーが移動する、アニメーション・プログラムを作成して下さい.

まず，先ほどのmap.cをmobile.cに名前を変えて保存しておいて下さい．これを改造していきます．

追加が必要な関数

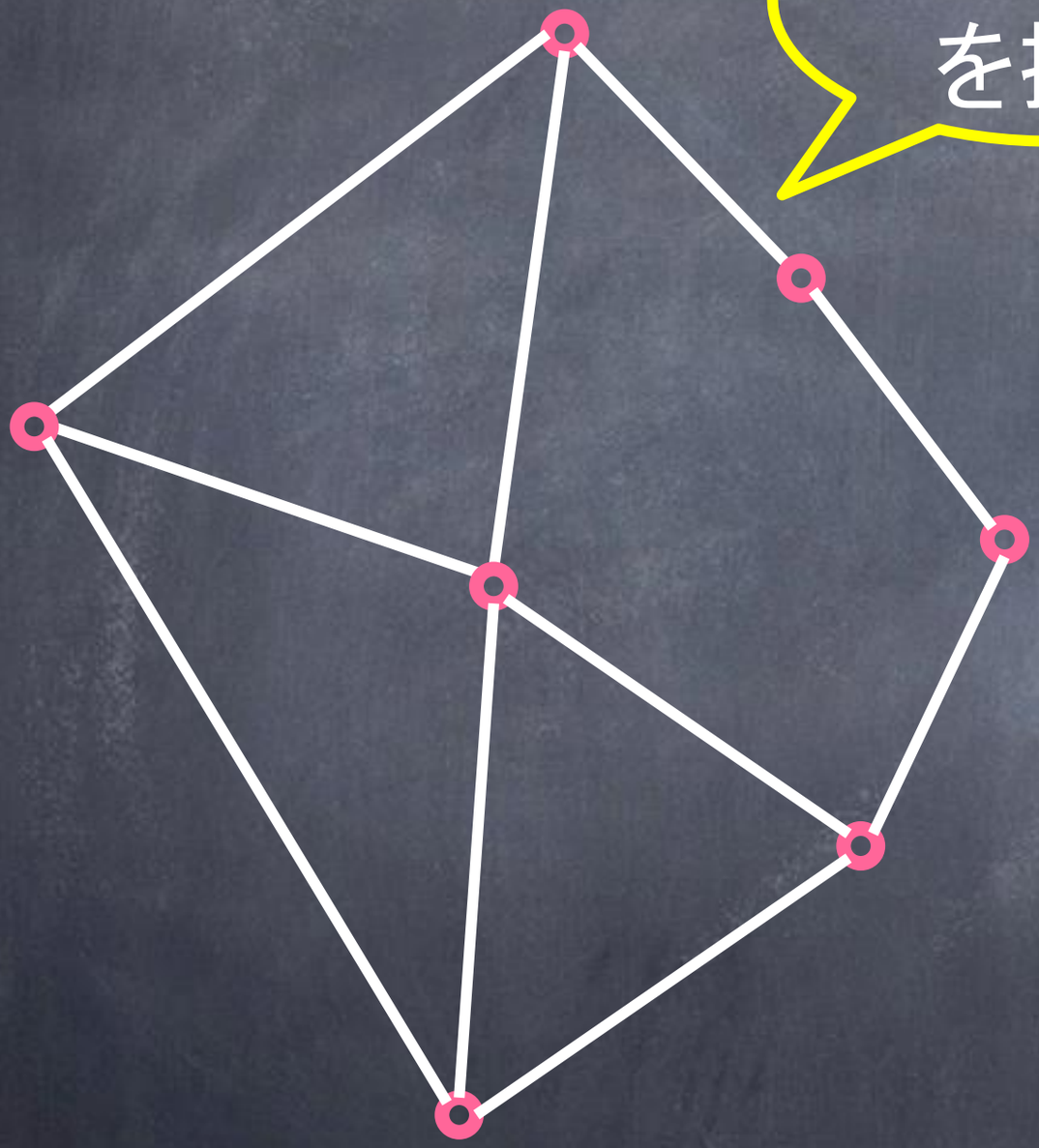
1. 目的地までの経路を決める関数
2.隣接する2つの交差点間をマーカが移動するアニメーション表示をする関数
3. 2の処理を目的地まで繰り返させる関数

コンピュータ実習（第10回）

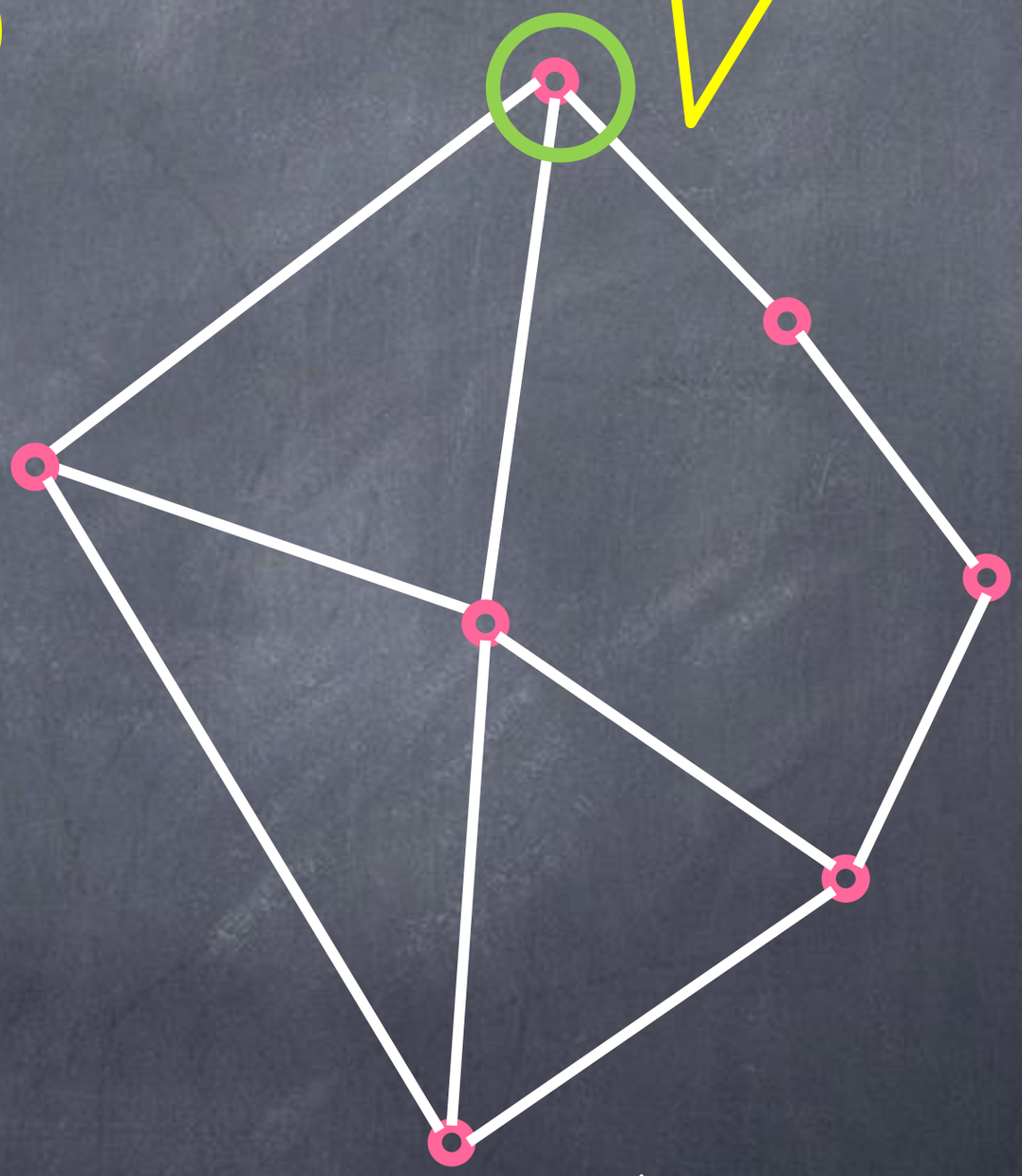
車の移動アニメーションのイメージ

1．地図
を描く

2．車を
描く



@フロントバッファ



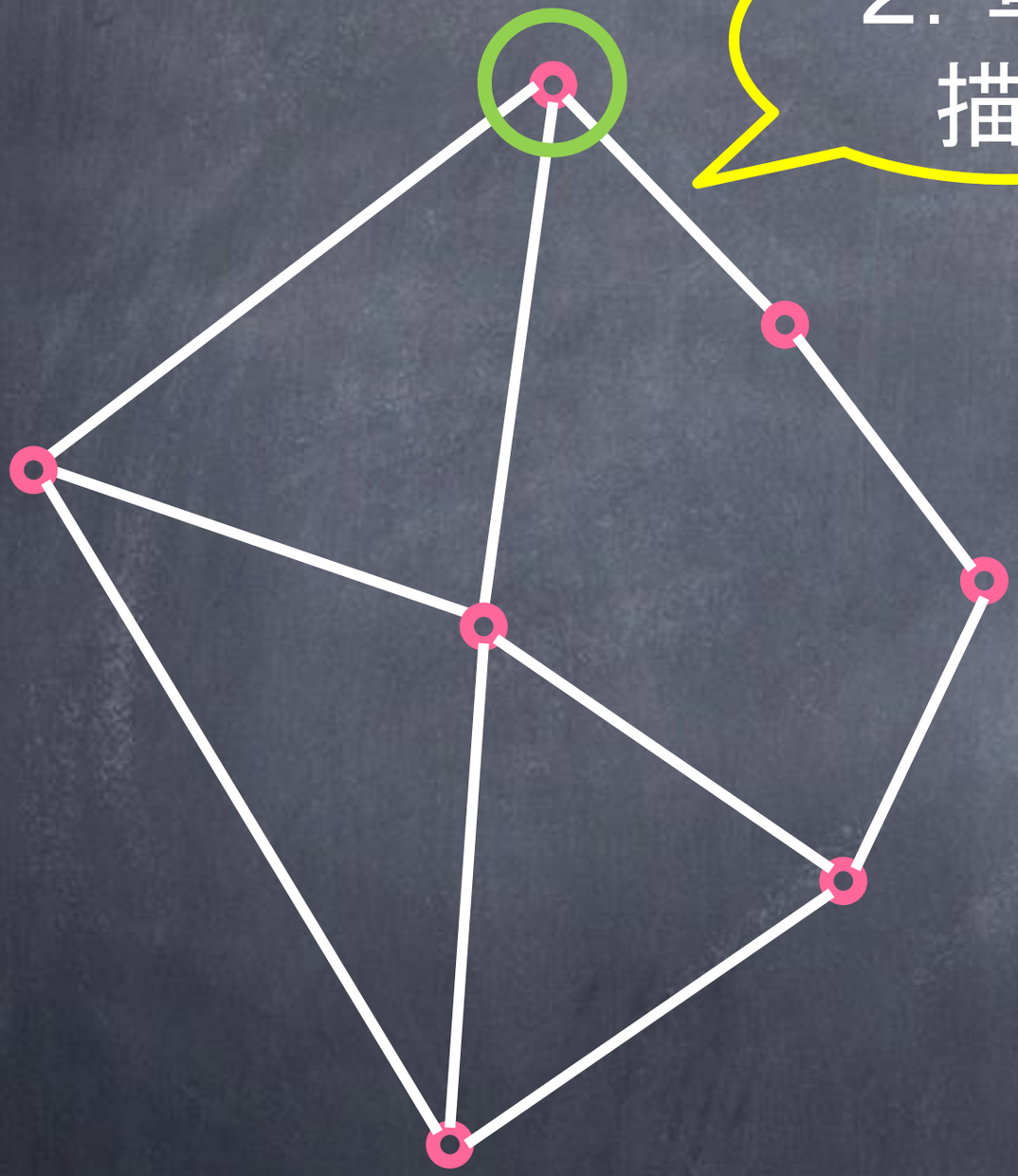
@フロントバッファ

コンピュータ実習（第10回）

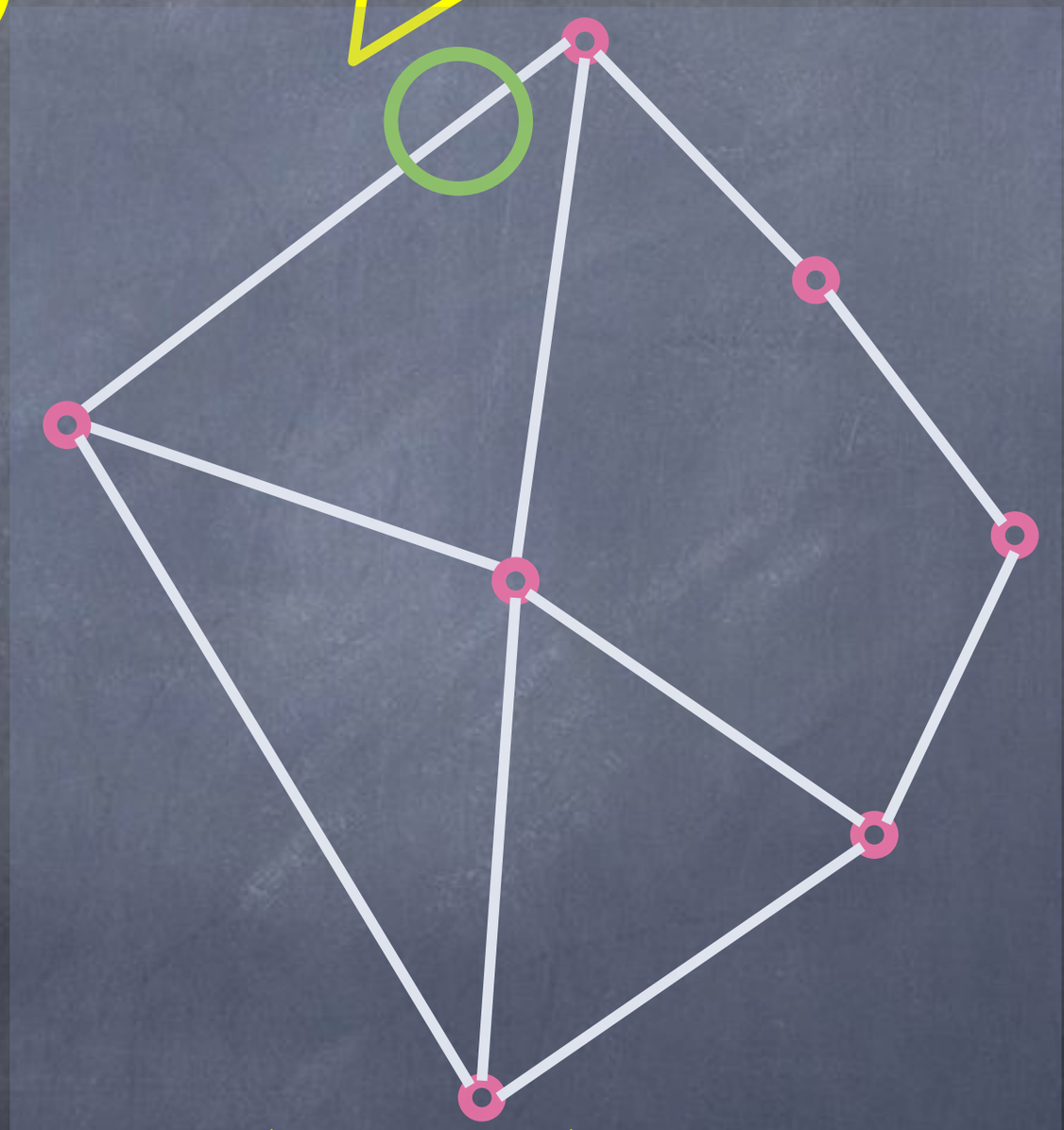
車の移動アニメーションのイメージ

2．車を
描く

3．地図＆少し
移動した車を描く



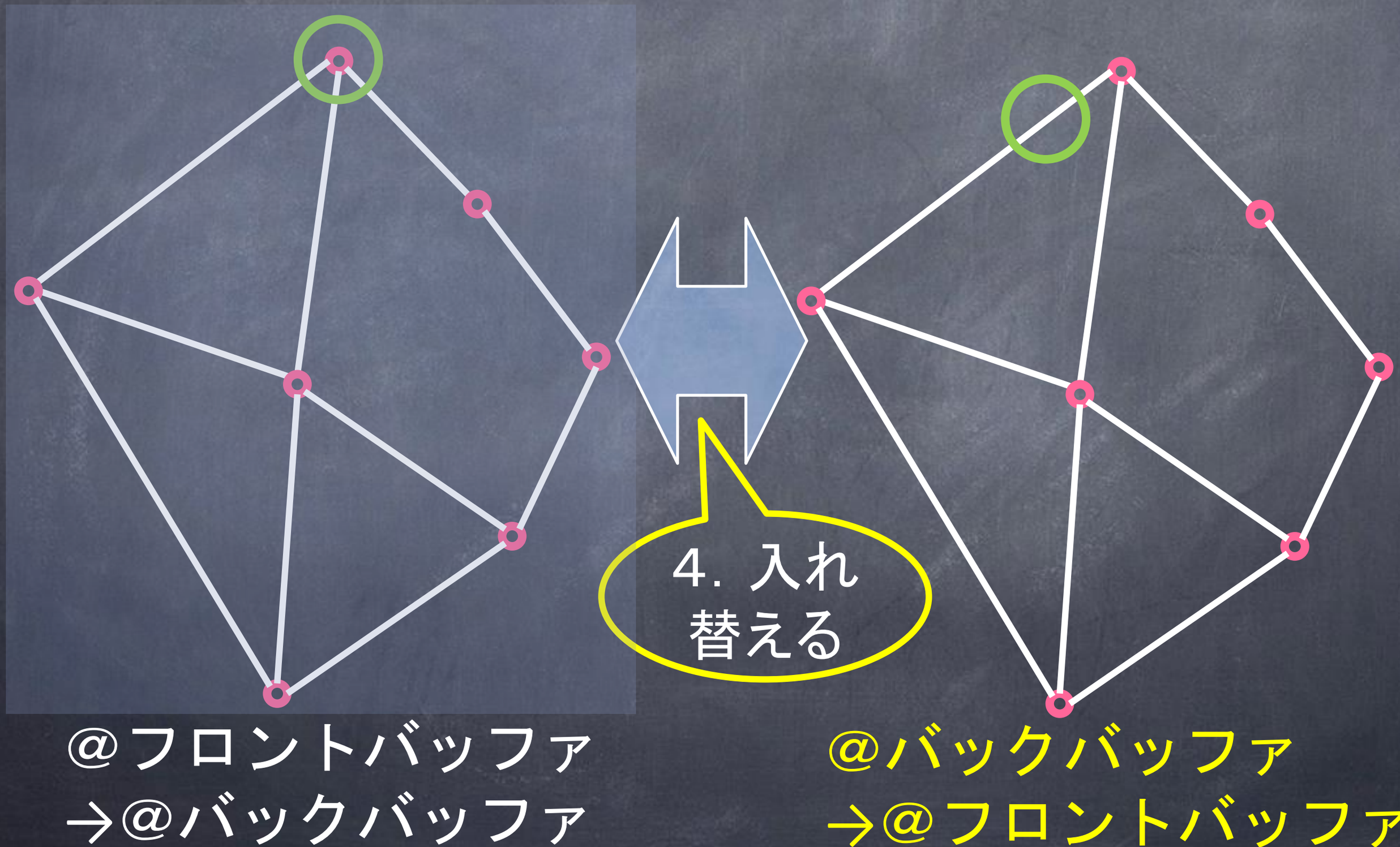
@フロントバッファ



@バックバッファ

コンピュータ実習（第10回）

車の移動アニメーションのイメージ



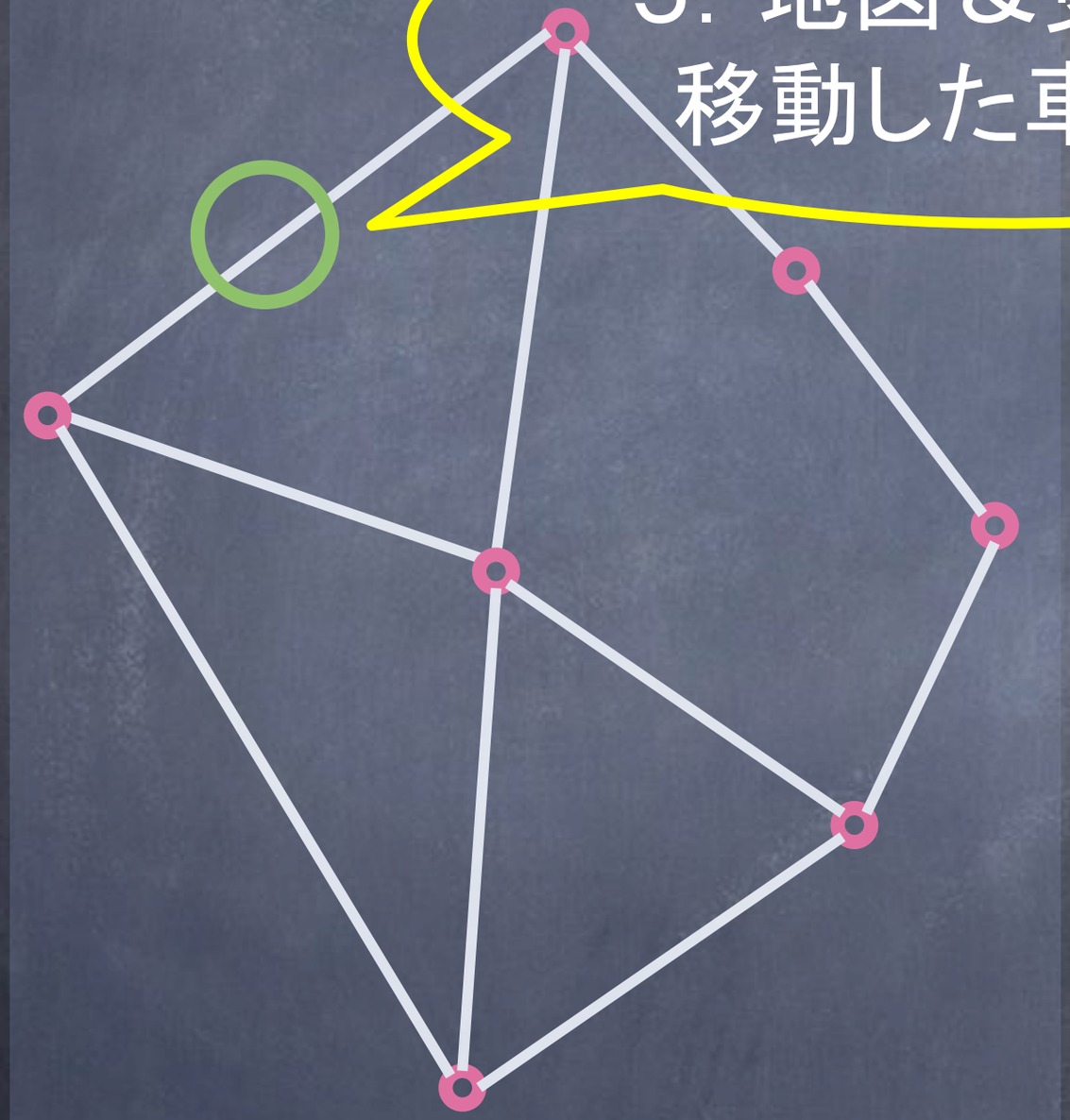
@フロントバッファ
→@バックバッファ

@バックバッファ
→@フロントバッファ

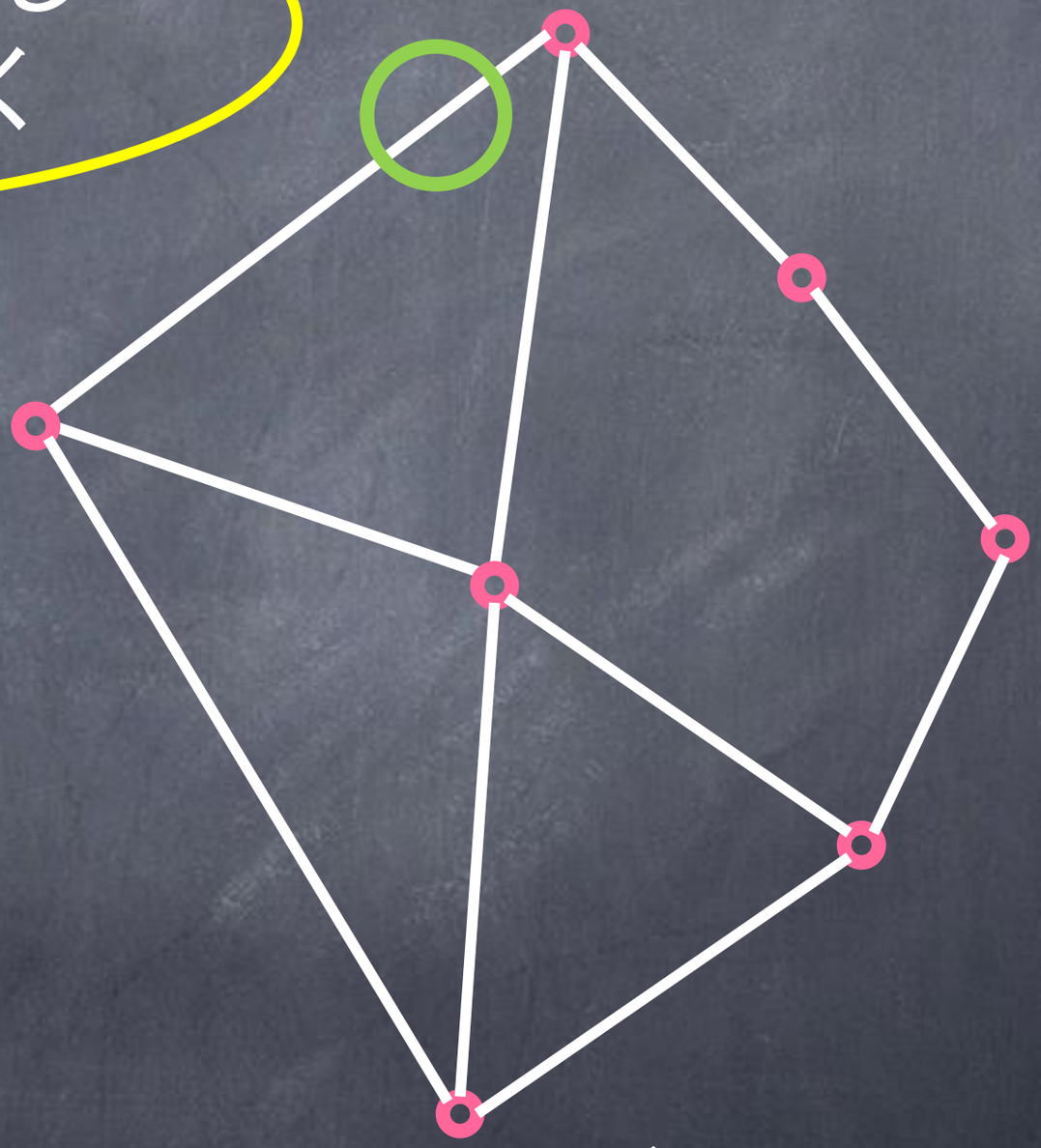
コンピュータ実習（第10回）

車の移動アニメーションのイメージ

5．地図＆更に少し
移動した車を描く



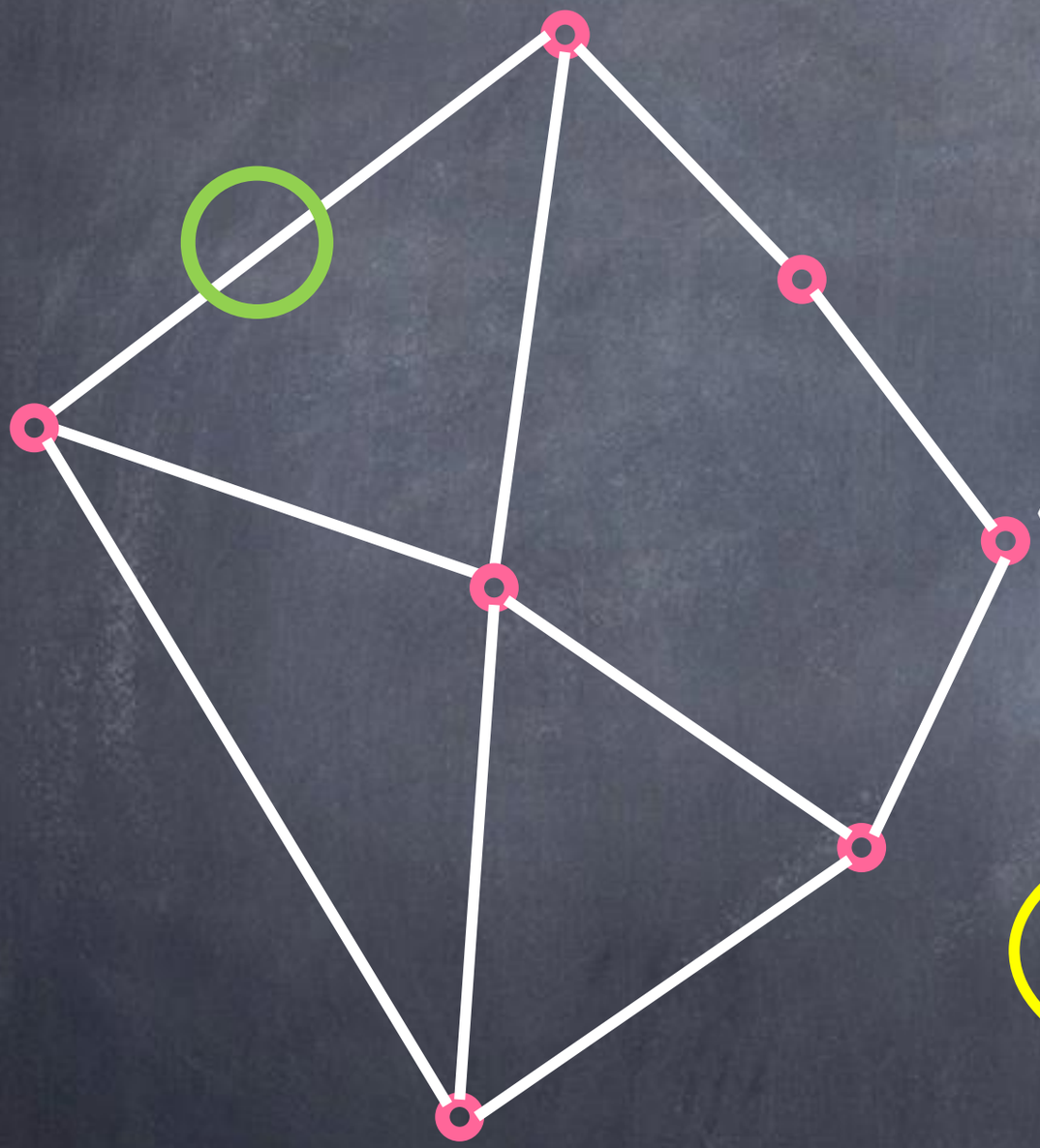
@バックバッファ



@フロントバッファ

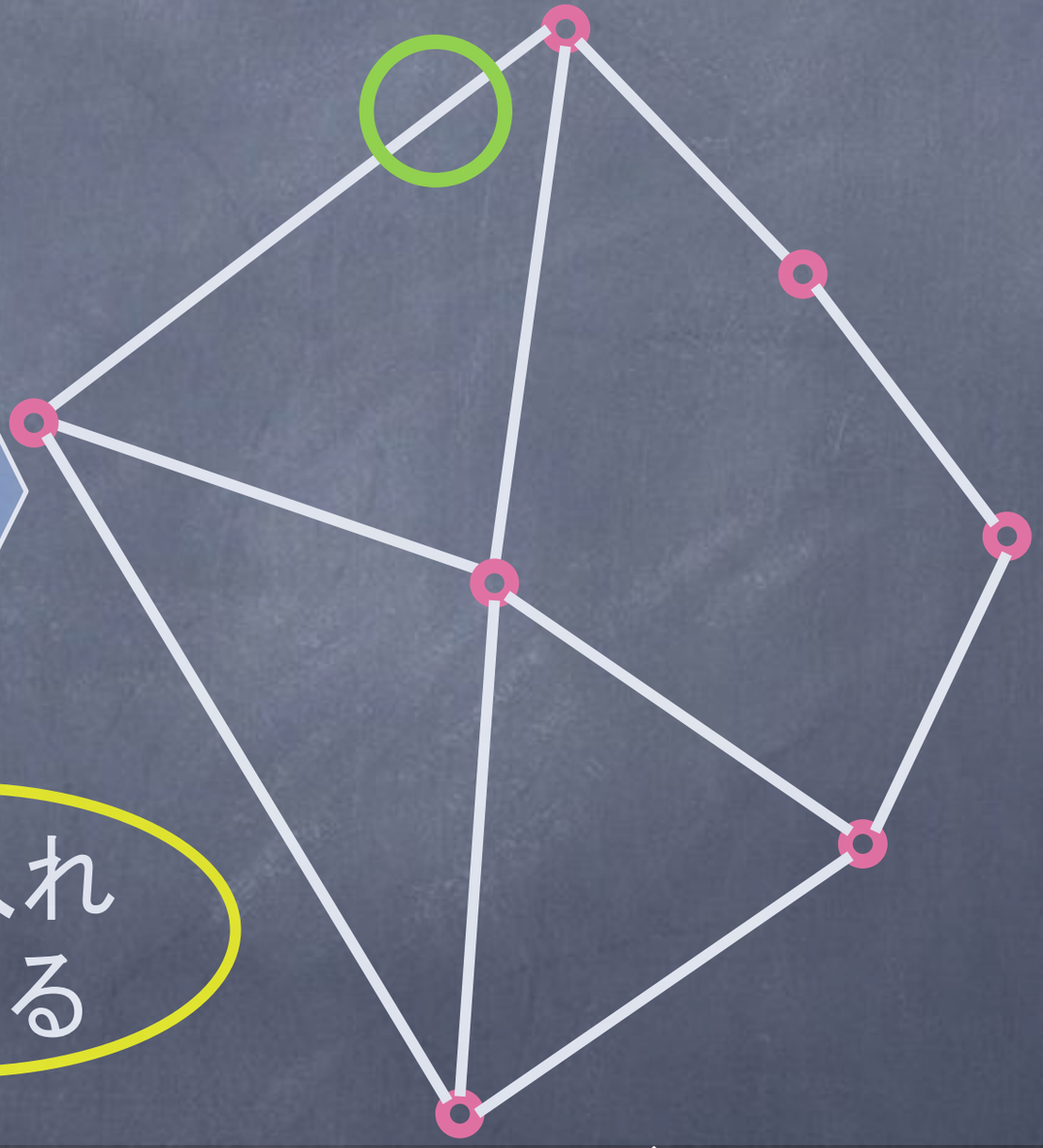
コンピュータ実習（第10回）

車の移動アニメーションのイメージ



@バックバッファ
→@フロントバッファ

6. 入れ
替える



@フロントバッファ
→@バックバッファ

コンピュータ実習（第10回）

第９回続き― 演習１ アニメーションによる可視化

予め与えた経路通りに、車を模したマーカが地図上を移動するプログラム

必要な関数・処理

1. 目的地までの経路を決める関数
   - 関数path_setとして与える（テキストの記載通り）
2. 隣接する２つの交差点間をマーカが移動するアニメーション表示をする
   - プログラムソースに書き込む
3. ２の処理を目的地まで繰り返させる関数
   - プログラムソースに書き込む

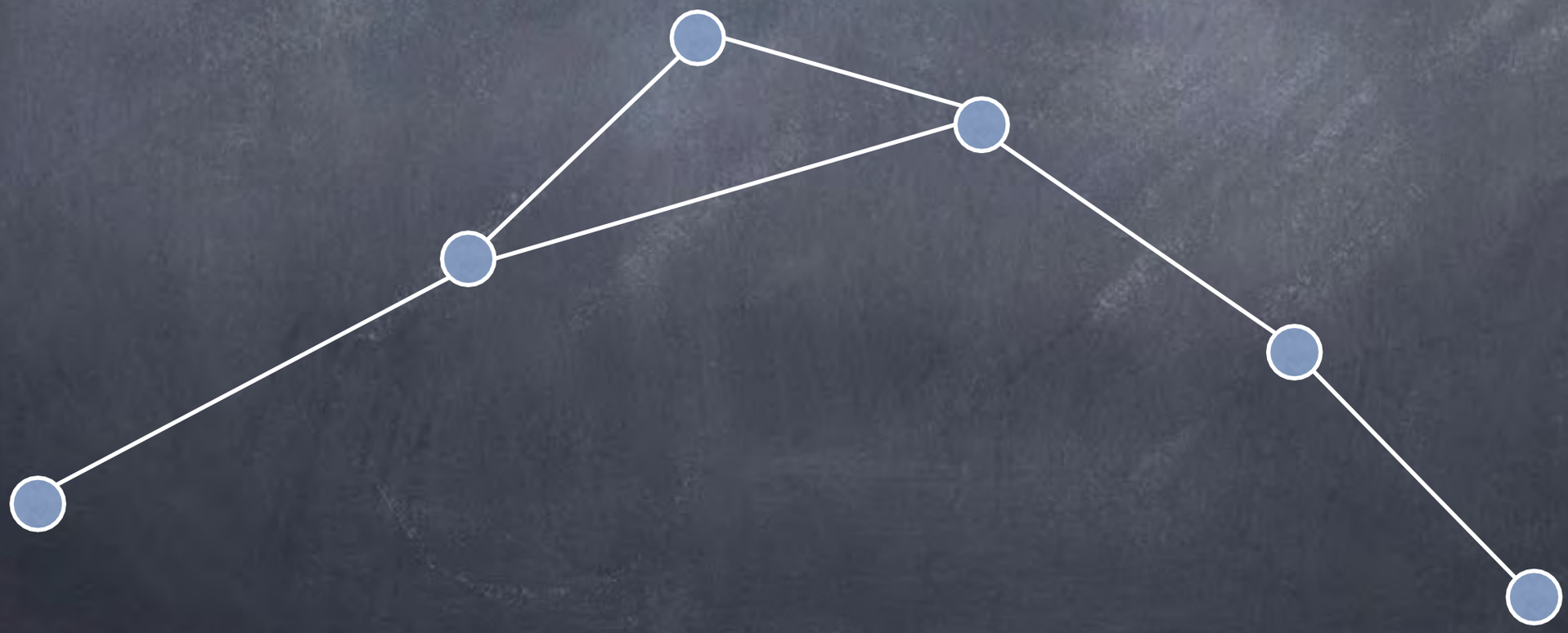
やりたいことは「地図上に現在地を示すマーカを描画する」こと！
そのために，現在地を経路上で移動させる必要がある.

コンピュータ実習（第10回）

演習1　アニメーションによる可視化

1. 目的地までの経路を決める関数　void path_set(void)

　　今回は予め自分で経路を決める



コンピュータ実習（第10回）

演習1　アニメーションによる可視化

1. 目的地までの経路を決める関数　void path_set(void)

今回は予め自分で経路を決める

隣接する交差点の設定(移動経路の設定)
交差点のID(数字)を配列 path[] に入れる。
例えば、経路上の交差点IDが: 1, 5, 3, 2, 7 だとすると、

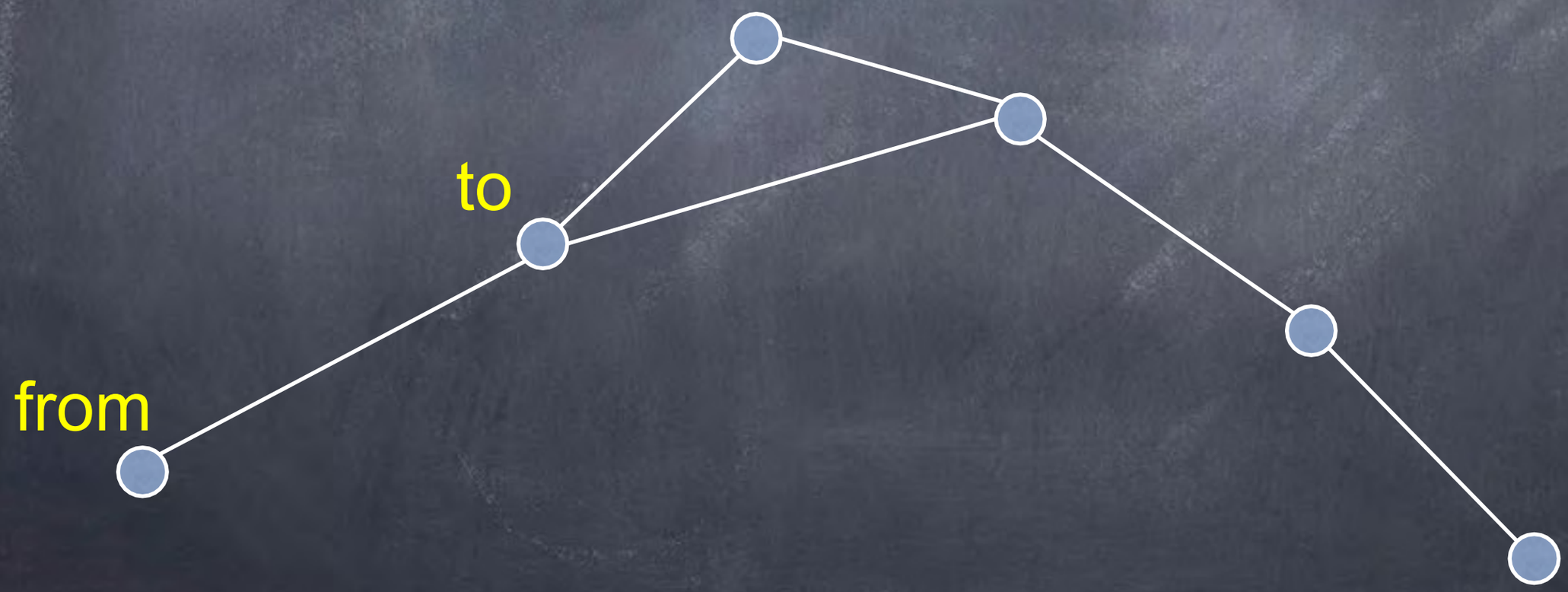
```
path[0] = 1;
path[1] = 5;
path[2] = 3;
path[3] = 2;
path[4] = 7;
path[5] = -1;
```

とする。最後の -1 は、経路が終ったことを示すダミーのID。

コンピュータ実習（第10回）

演習1　アニメーションによる可視化

2. 隣接する2つの交差点間をマーカが移動する
　　アニメーション表示をする



コンピュータ実習（第10回）

演習１　アニメーションによる可視化

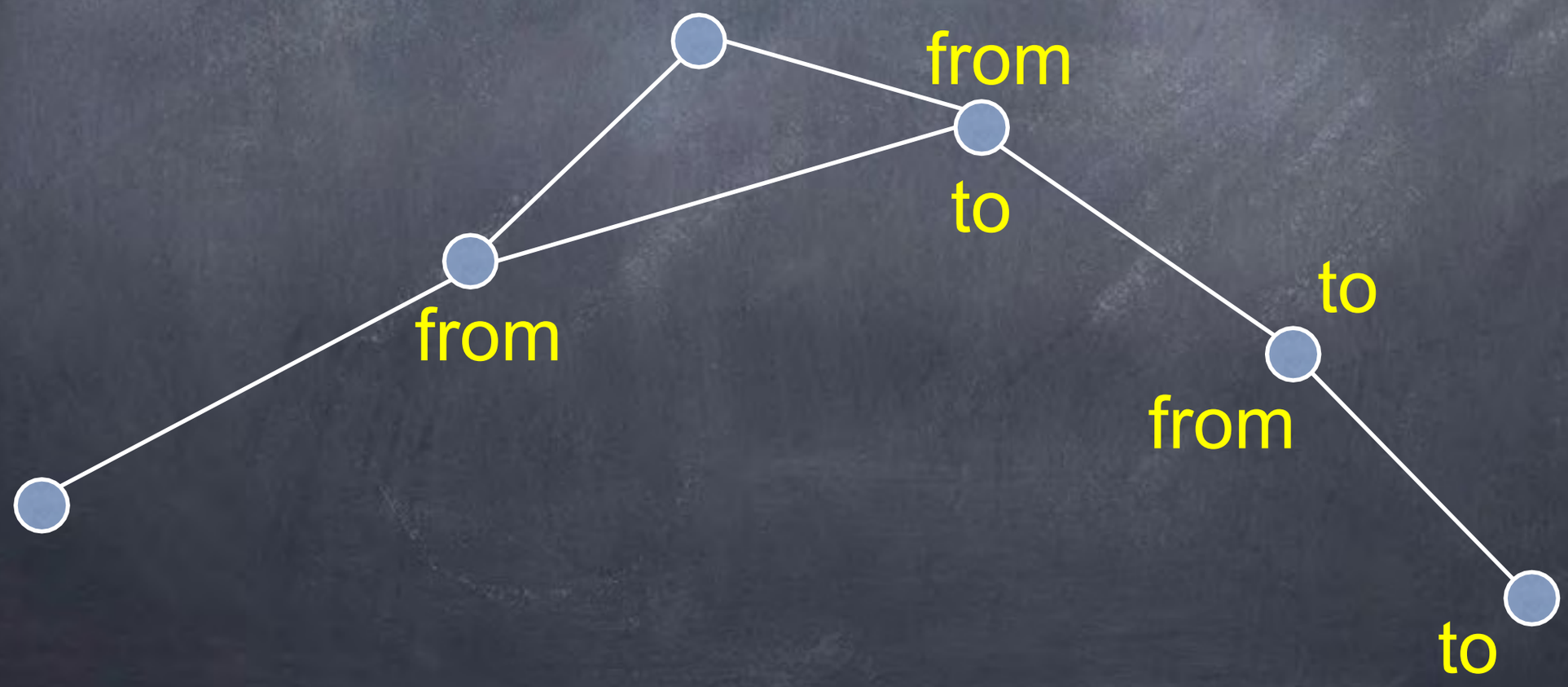
2. 隣接する2つの交差点間をマーカが移動する
アニメーション表示をする

```
/* 現在の交差点と次の交差点を (x0, y0),(x1, y1)とする */
        x0 = (現在の交差点のx座標)
        y0 = (現在の交差点のy座標)
        x1 = (次の交差点のx座標)
        y1 = (次の交差点のy座標)
        distance = hypot(x1 - x0, y1 - y0);  /* 2点間の距離を計算  */
        steps = (int)(distance / 0.1);  /* 距離に応じてステップ数を決める */
/* 交差点の間で現在の位置(vehicle_x, vehicle_y)を少しずつ進める */
        vehicle_stepOnEdge++;
        vehicle_x = (車両マーカーのx座標位置の更新)
        vehicle_y = (車両マーカーのy座標位置の更新)
/* 現在の位置に移動体（円形）を表示 */
        glColor3d(1.0, 1.0, 1.0);
        draw_circle(vehicle_x, vehicle_y, MARKER_RADIUS);
```

コンピュータ実習（第10回）

演習１　アニメーションによる可視化

３．２の処理を目的地まで繰り返させる



コンピュータ実習（第10回）

演習１　アニメーションによる可視化

３．２の処理を目的地まで繰り返させる

```
/* 今の交差点と次の交差点のIDがどちらもダミーでない時 */
if (path[ /* パス上の現在の交差点を表すインデクス */ ] != -1 &&
    path[ /* パス上の次の交差点を表すインデクス */ ] != -1) {
    /* まだゴールに達していないので、移動体の位置を進める */
    /* 現在の交差点と次の交差点を (x0, y0),(x1, y1)とする */
    /* 交差点の間で現在の位置(vehicle_x, vehicle_y)を少しずつ進める */

      ２のアルゴリズムがここに入る！！！

    if(/* 現在の位置が次の交差点を過ぎていたら */ ){
        /* パス上の現在の交差点を表すインデクスを進める */
    }
}
/* 現在の位置に移動体を表示 */
```

コンピュータ実習（第10回）

演習2　サーチを利用したマーカの移動

出発地と目的地を指定するとその間の経路(上の交差点)を見つけ出す簡単なプログラムを作成してください.

まず，演習1で作成したmobile.cをmobile_s.cに名前を変えて保存しておいて下さい．これを改造していきます.

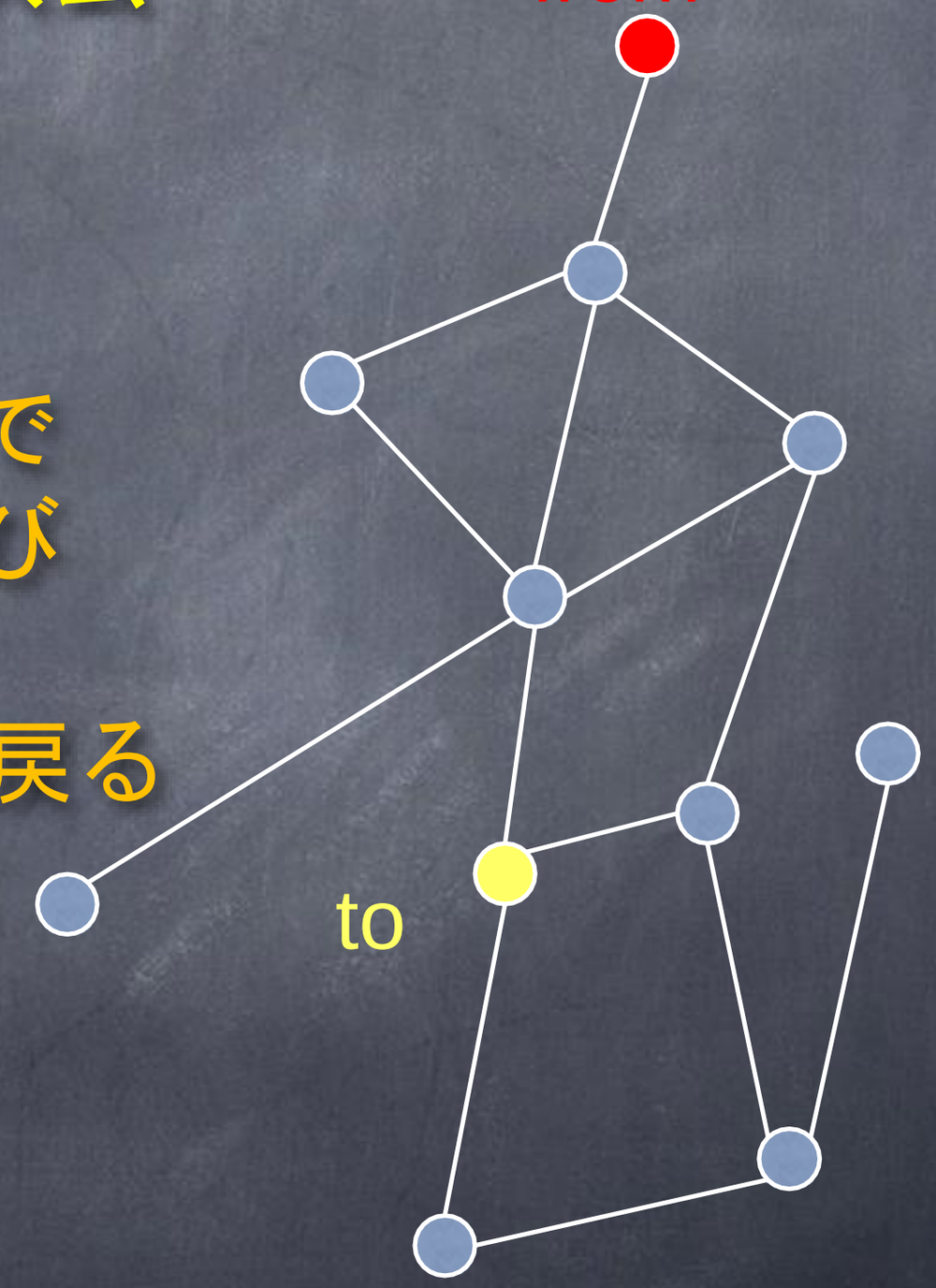
演習1では経路をプログラム中で指定しました.

一連の実習で最終的に使用するサーチアルゴリズムは次回の演習で勉強しますので，
今回の演習では一番簡単なサーチアルゴリズムをコーディングします.

コンピュータ実習（第10回）

第9回続き― 演習2 サーチを利用したマーカの移動

最近傍隣接交差点検索アルゴリズム



必要な関数・処理

1. 現在地を出発点として
2. 出発地に隣接する交差点の中で
   最も目的地に近い交差点を選び
3. そこに移動して
4. そこが目的地でなければ１に戻る
5. 目的地であれば探索終了

まずはこれを使って演習２を完成させて下さい

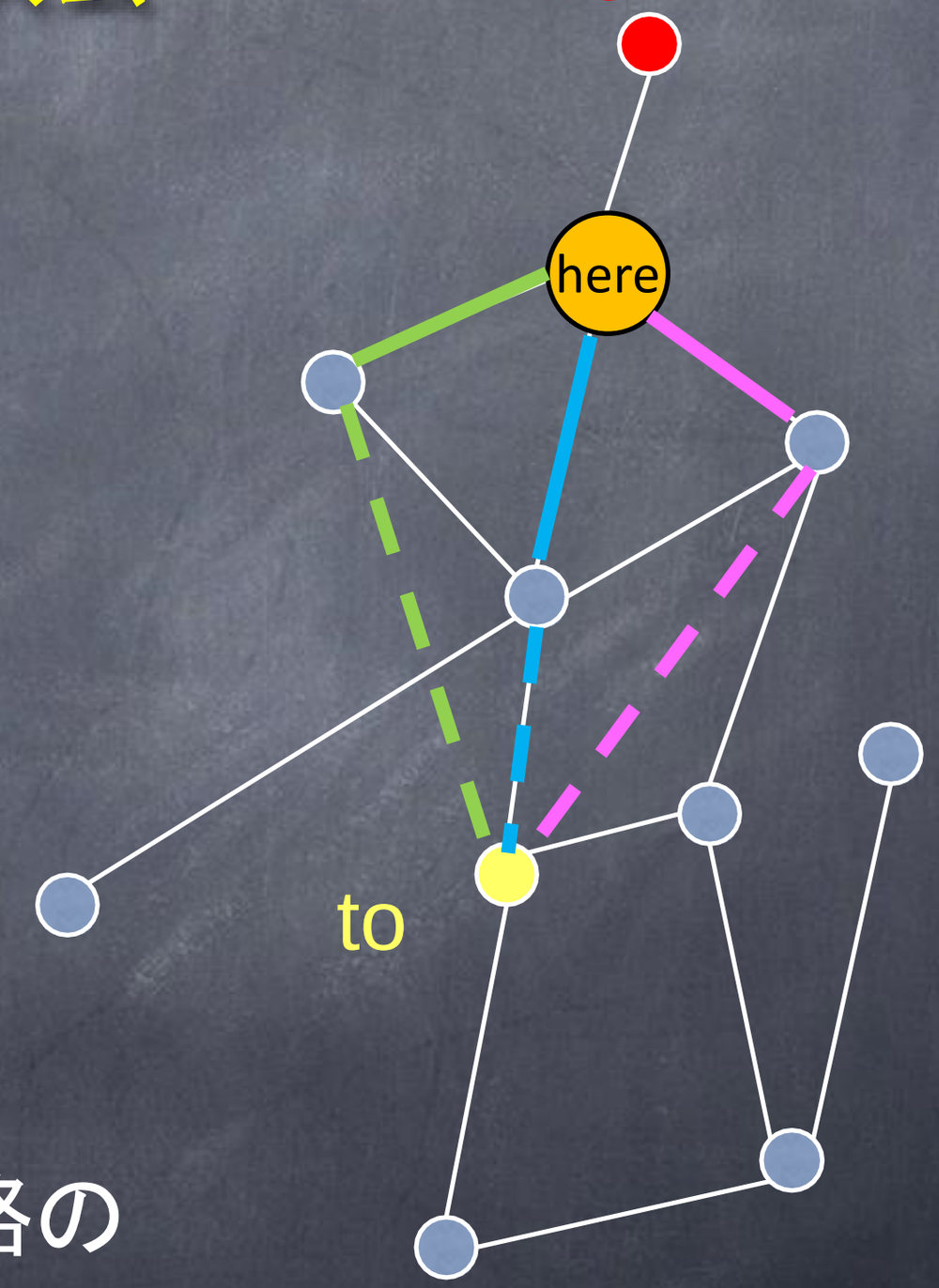
コンピュータ実習（第10回）

第9回続き― 演習2 サーチを利用したマーカの移動

最近傍隣接交差点検索アルゴリズム

この場合のサーチの考え方

1. 隣接交差点から目的地までの直線距離を計算する
2. 隣接交差点から目的地までの直線距離が最も短い交差点を選択する
3. その交差点に移動する



とりあえず、なんとなく最短経路の探索が出来る気がしませんか・・・？

コンピュータ実習（第10回）

演習2　サーチを利用したマーカの移動

最近傍隣接交差点検索アルゴリズム

目的地に最も近い隣接交差点のサーチアルゴリズム

現在地交差点ID : id と 目的地交差点ID : goal
が与えられているとき、goal に最も近い隣接交差点の id は、次のようにして求められる。

```
min_distance = 1e100; /* 最短距離を暫定的に大きな値としておく */
for (i = 0; i < cross[id].points; i++)
    {
        distance = (cross[goal] と cross[id]のi番目の隣接交差点との距離を計算);
       /* もしこれまでの最短距離よりも短いものが見つかったら */
       if (min_distance > distance)
          {
               min_distance = distance;     /* 新たな最短距離をセット */
               nearest = cross[id].next[i];   /* 最も近い隣接交差点IDを更新 */
           }
    }
```

コンピュータ実習（第10回）

演習2　サーチを利用したマーカの移動

最近傍隣接交差点検索アルゴリズム

「交差点毎に次の隣接交差点を調べてそこに移動する」という一番素直なやり方

ヒント

- **int search_nearest(int id, int goal) の追加**
  - 交差点 id の隣接交差点の中から交差点 goalに 最も近いものを探し、return文により返す
- **void path_set(void) の削除**
  - この関数はもう要りません
- **int main(void) の修正**

弱点

同じ経路を行ったり来たりする場合がある

→行き止まりや袋小路

コンピュータ実習（第10回）

演習2　サーチを利用したマーカの移動
最近傍隣接交差点検索アルゴリズム

同じ経路を行ったり来たりする袋小路問題の解決策を考える
　例えば、
　　さっきいた交差点は計算の際の候補に加えない
　　今まで通ってきた交差点は計算の際の候補に加えない
　など、が考えられます。

しかし、これだけでは不十分ですので、考えてみて下さい。

12/14の授業では、この問題を解決可能な最適経路探索アルゴリズムとして「ダイクストラ法」を取り上げます。

コンピュータ実習（第10回）

最短経路探索

今日の目標

グラフィックのことは置いておいて，最短経路を求める方法を考えてみる.

- 最近房隣接交差点探索 (先週に引き続いて)
- ダイクストラ法による最短経路探索

※今日の授業でOpenGLは使いません

コンピュータ実習（第10回）

最短経路探索

さきほどの最近傍隣接交差点検索アルゴリズム

1. 現在地を出発点とする

2. 出発点に隣接する交差点の中で最も目的地に近い交差点を選ぶ

3. 選んだ交差点に移動

4. 移動先が目的地でなければ1に戻る

5. 目的地であったら終了

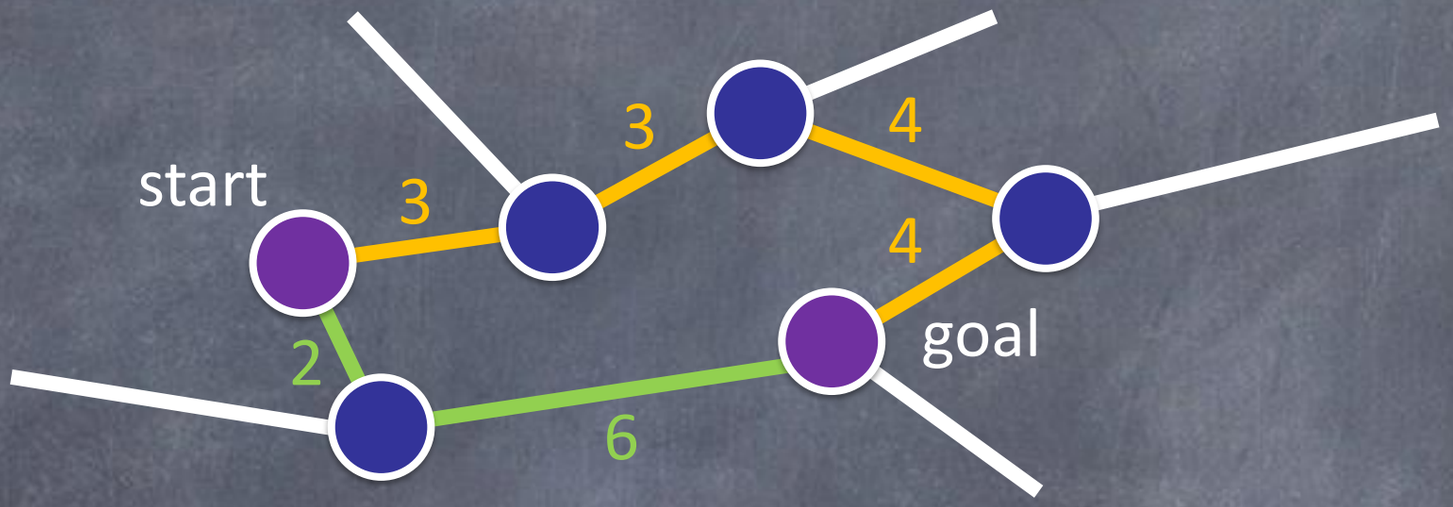
このアルゴリズムには，
致命的な欠陥があります

コンピュータ実習（第10回）

最短経路探索

最近房隣接交差点検索アルゴリズムの欠陥

- 必ずしも最短経路が選ばれるわけではない



- 目的地にたどり着けない場合がある

コンピュータ実習（第10回）

最短経路探索

最近傍隣接交差点検索アルゴリズムの改善策

- 1つ前にいた交差点は計算（候補）から除外

- 今まで通ってきた交差点は計算（候補）から除外

- 次に行く交差点が隣接交差点を一つしか持たない（＝行き止まり）の場合はその交差点を計算（候補）から除外

- etc.

コンピュータ実習（第10回）

最短経路探索

コンピュータ実習の単位取得条件

- 「カーナビゲーションプログラムを完成させること」
  - 出発地・目的地のキーボードからの入力
  - 交差点データの構造体への読み込み
  - 構造体からのデータの読み出し
  - ターミナルへのテキスト表示
  - 最低限の地図およびマーカ移動グラフィックス表示
  - 経路探索の実装（目的地までたどり着くこと）
    - 方法１：改良した最近傍隣接交差点検索
    - 方法２：ダイクストラ法

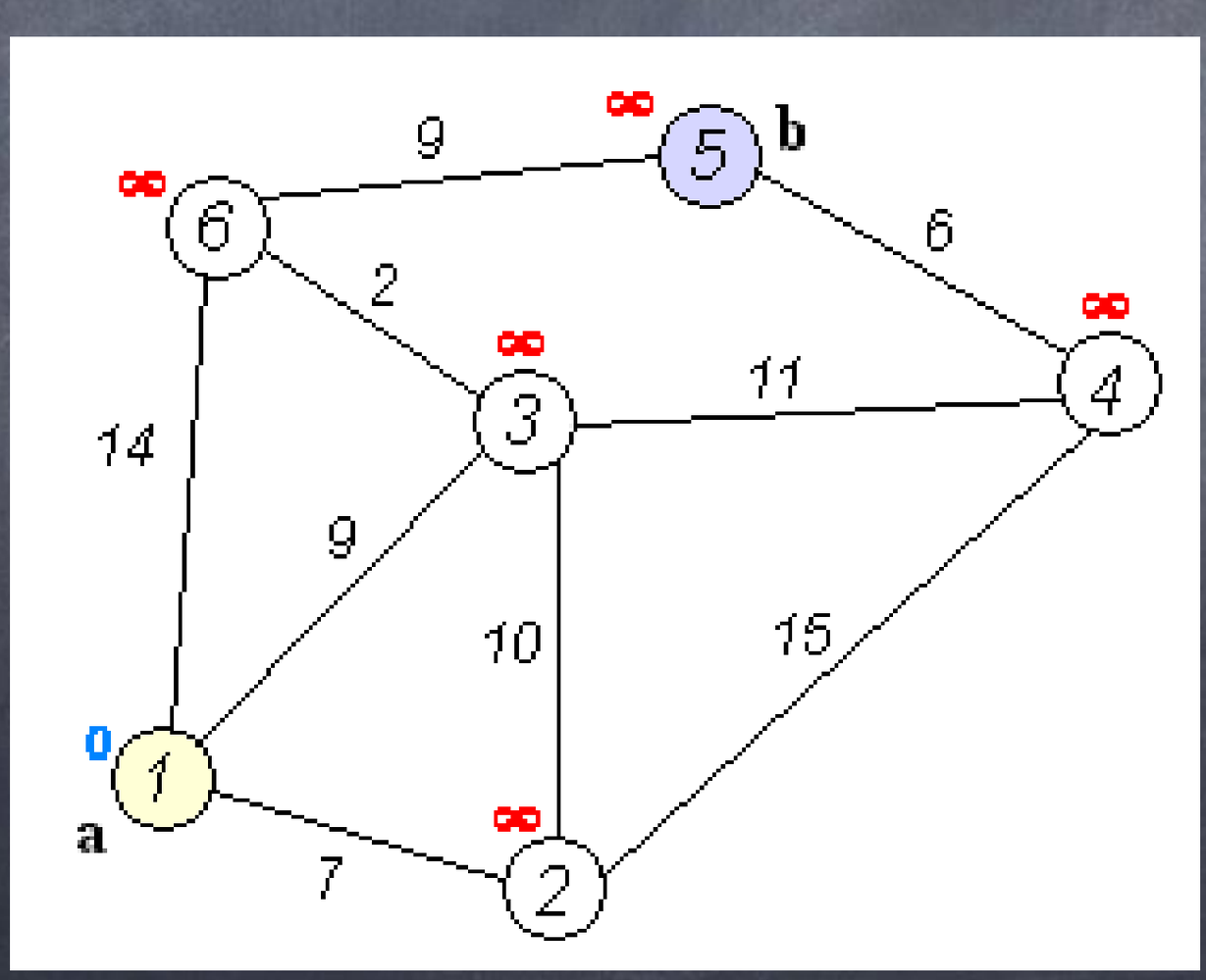
今回が最低限のプログラムを作る上で必要な最終過程

コンピュータ実習（第10回）

最短経路探索

ダイクストラ法のアルゴリズム

- グラフ理論における最短経路問題を効率的に解くためのアルゴリズム
- 道路網をグラフとして表現する（交差点：ノード，道：枝）



(wikipediaより引用)

コンピュータ実習（第10回）

最短経路探索

ダイクストラ法のアルゴリズム

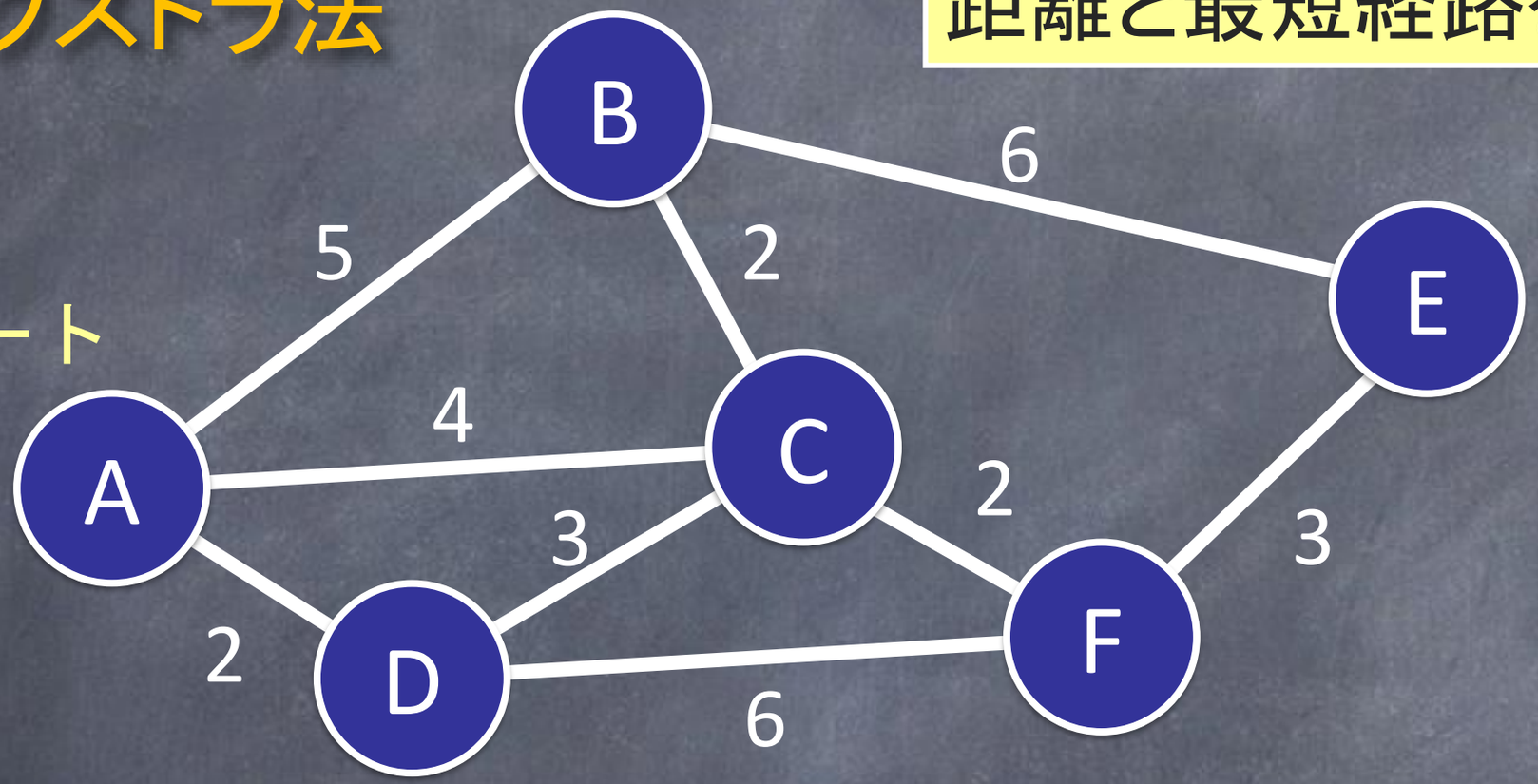
- 道路網をグラフとして表現する（交差点：ノード，道：枝）
- 各交差点（ノード）にいくらのコスト（距離）でたどり着けるのかを計算する.
- アルゴリズム

①始点となるノード（節点）を決定（出発点）
②始点となるノード以外の点を、未定義(または無限大)に設定
③始点のノードに隣接する全ノードの距離を求める
④最短距離のノードを確定する
⑤確定したノードに隣接している全ノードの距離を求める
　- 新しく計算された距離が現在の（ノードが保持する）距離より短い
　⇒　新しく計算された距離を、確定距離とする
　- それ以外　⇒　現在の距離を、そのまま確定距離としておく
⑥確定していないノードのうち始点から最短距離のノードを確定する
⑦全ノードが確定されるまで、操作⑤～⑥を繰り返す

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法

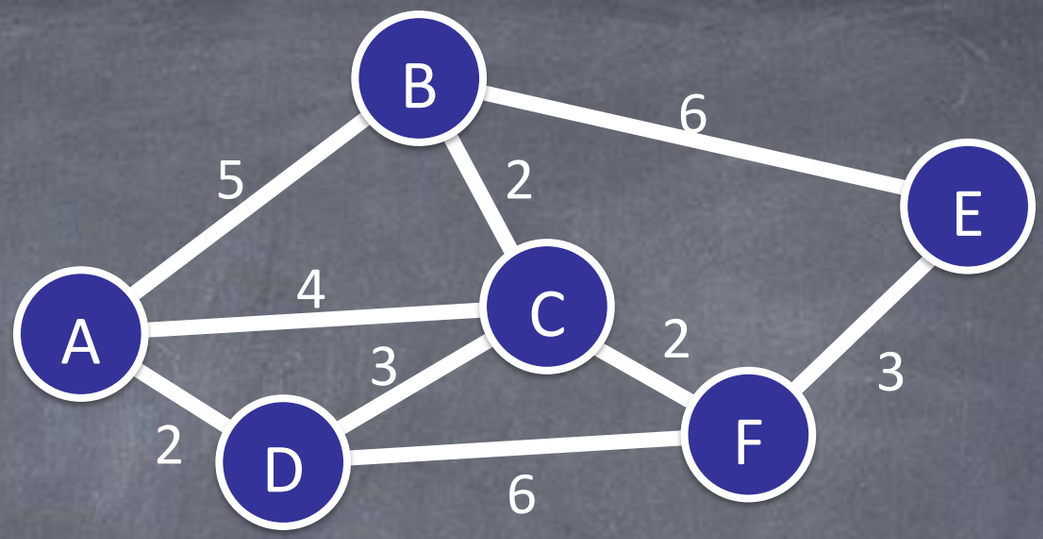
ノードAから各ノードへの最短距離と最短経路を求める

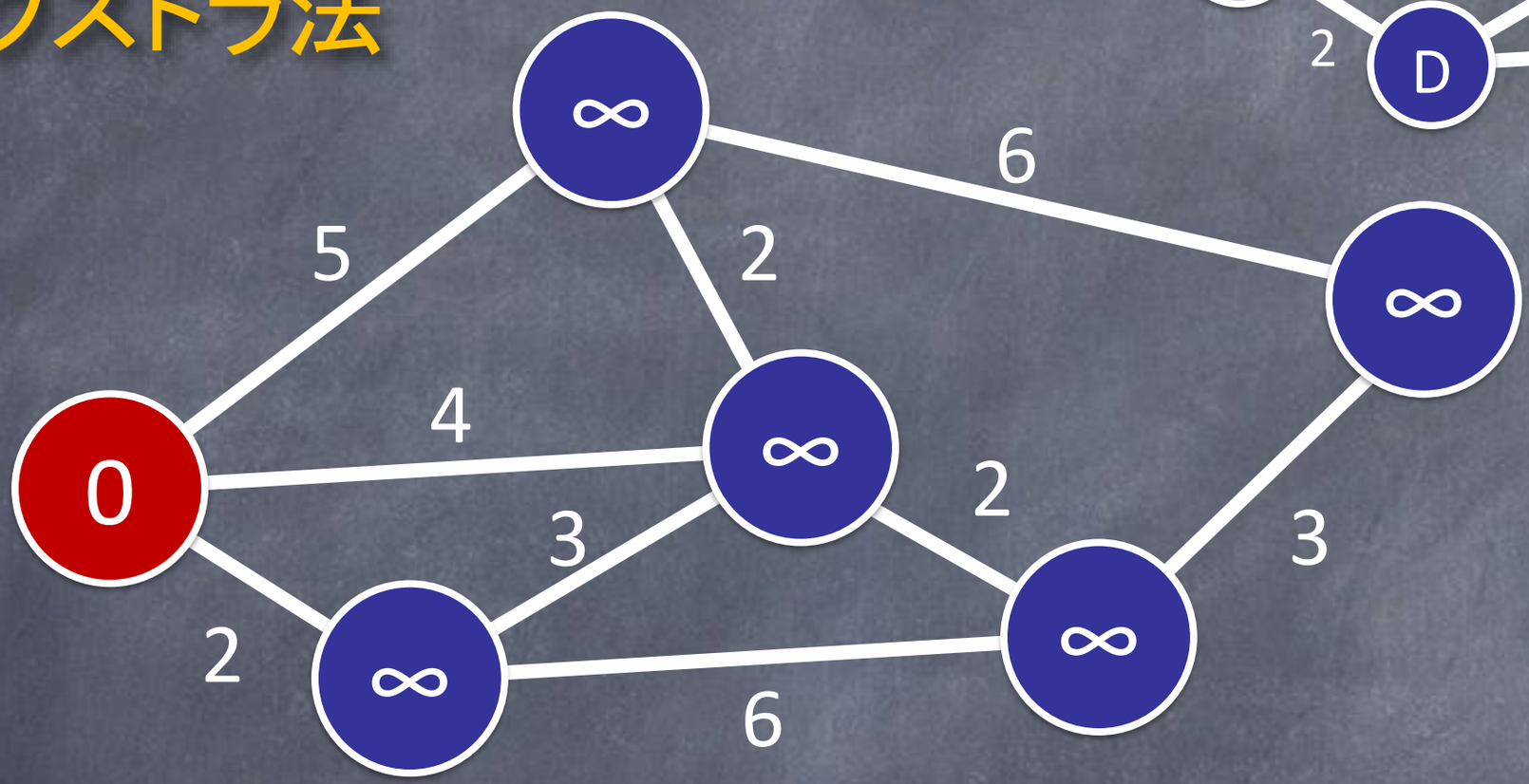


交差点	最短距離	最短経路
A	0	A
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法



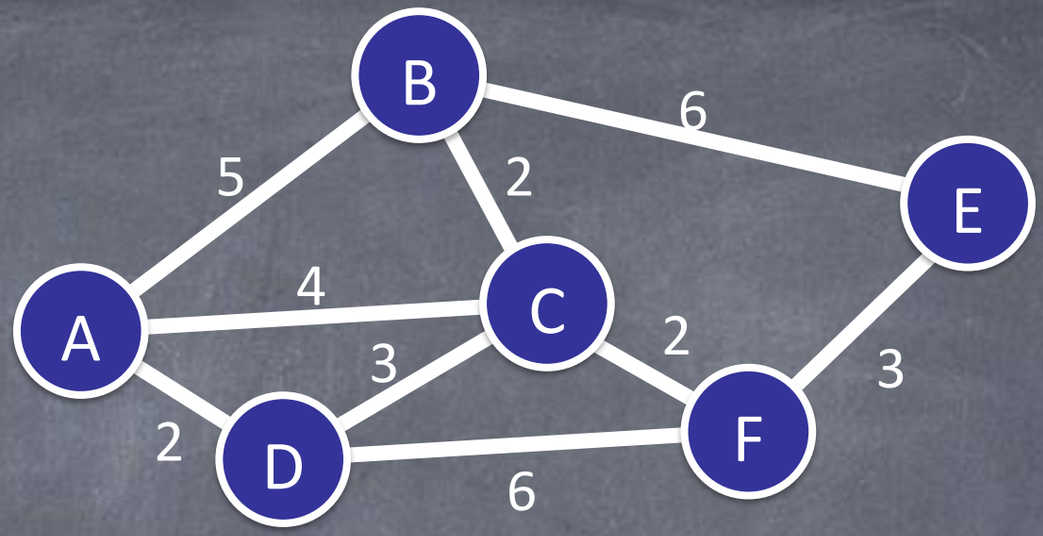


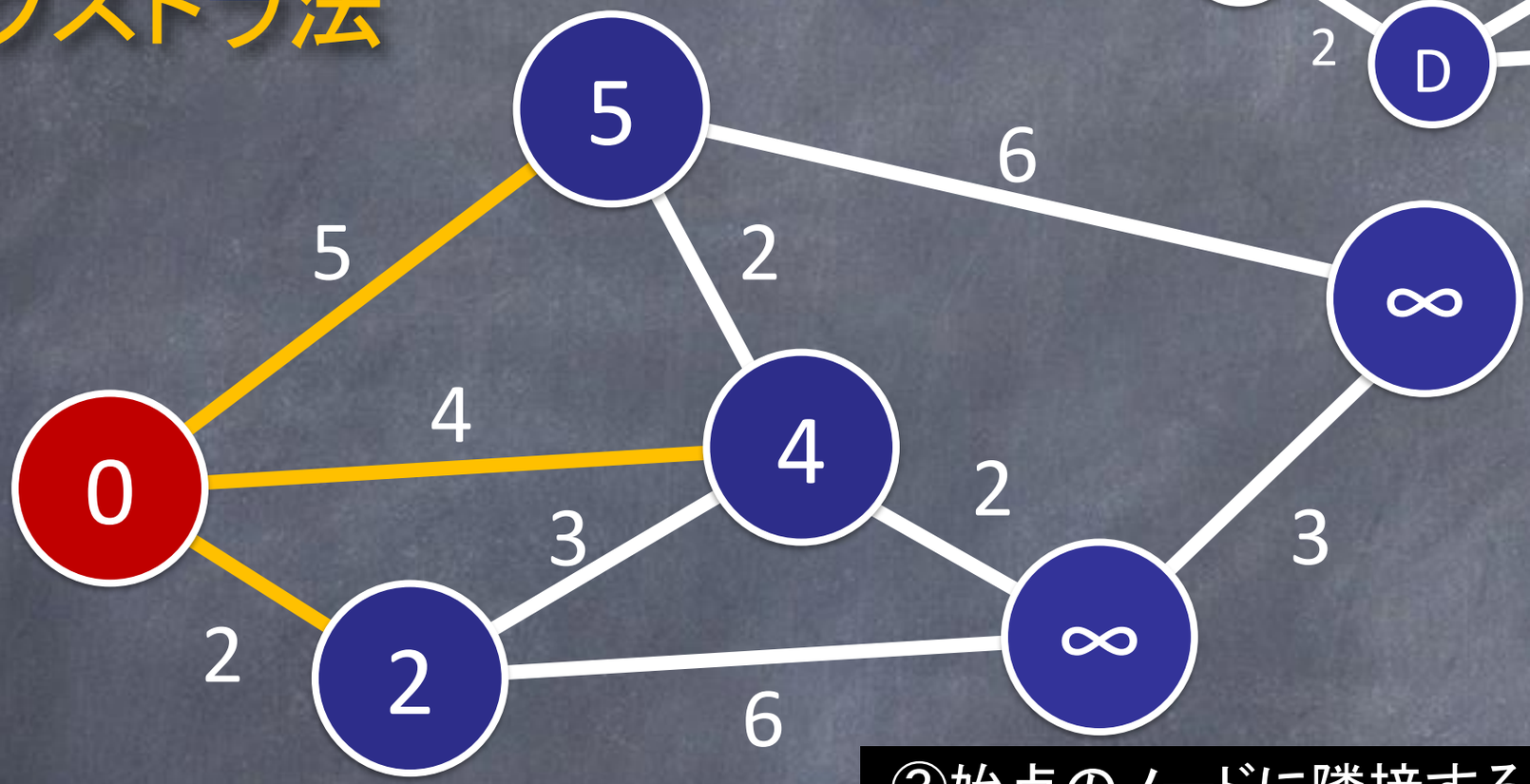
交差点	最短距離	最短経路
A	0	A
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	

①始点となるノード（節点）を決定（出発点）
②始点となるノード以外の点を、未定義(または無限大)に設定

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法



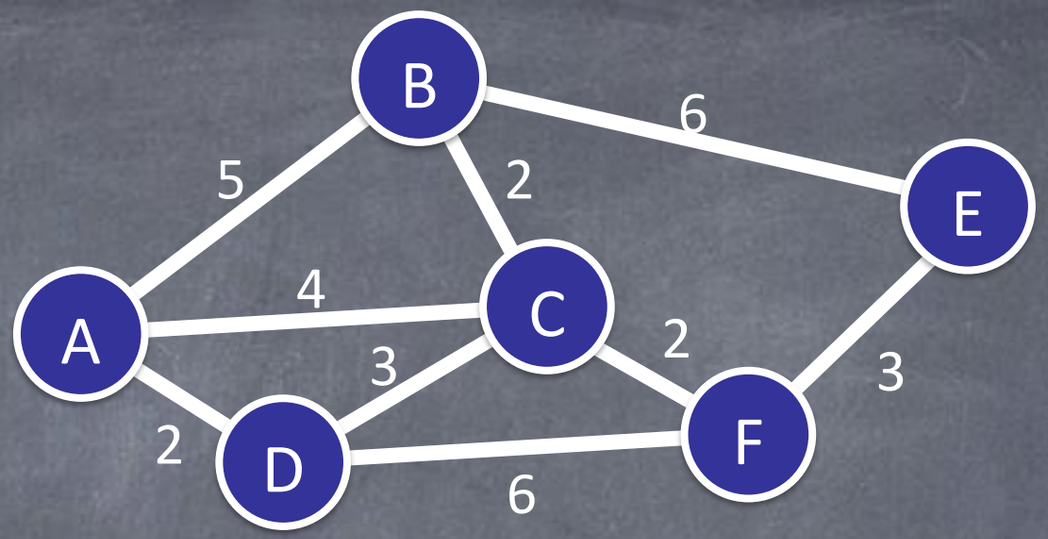


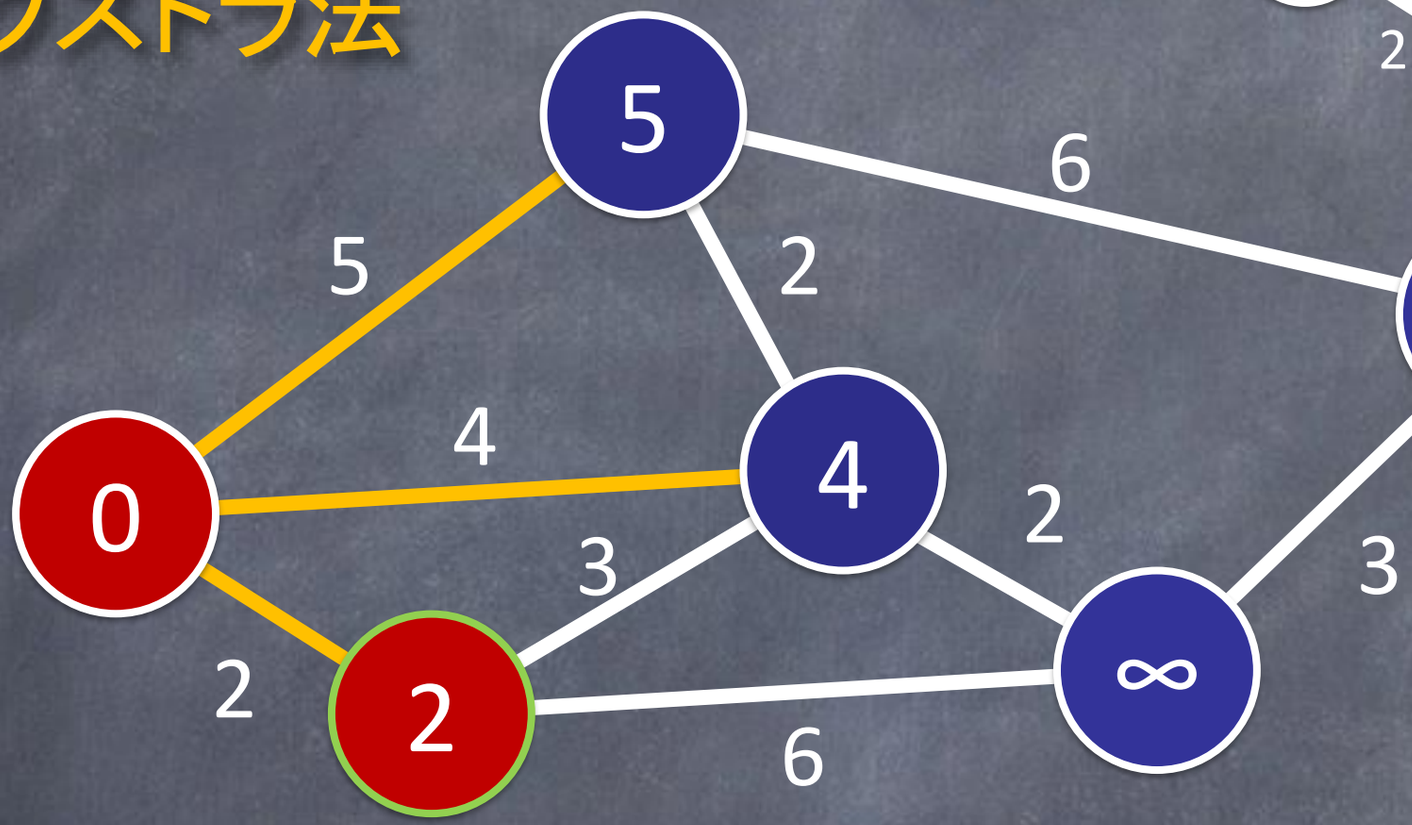
③始点のノードに隣接する全ノードの距離を求める

交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	∞	
F	∞	

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法



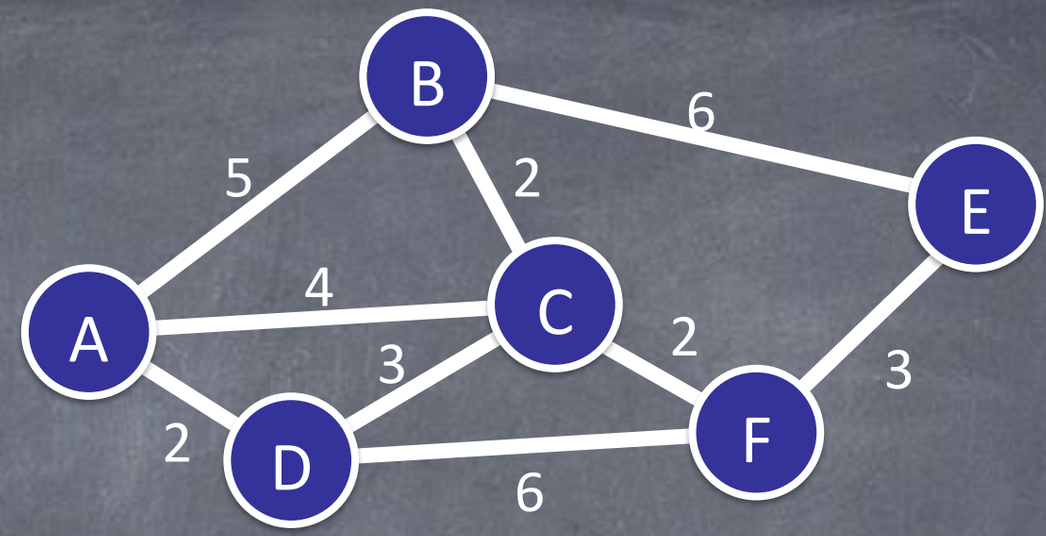


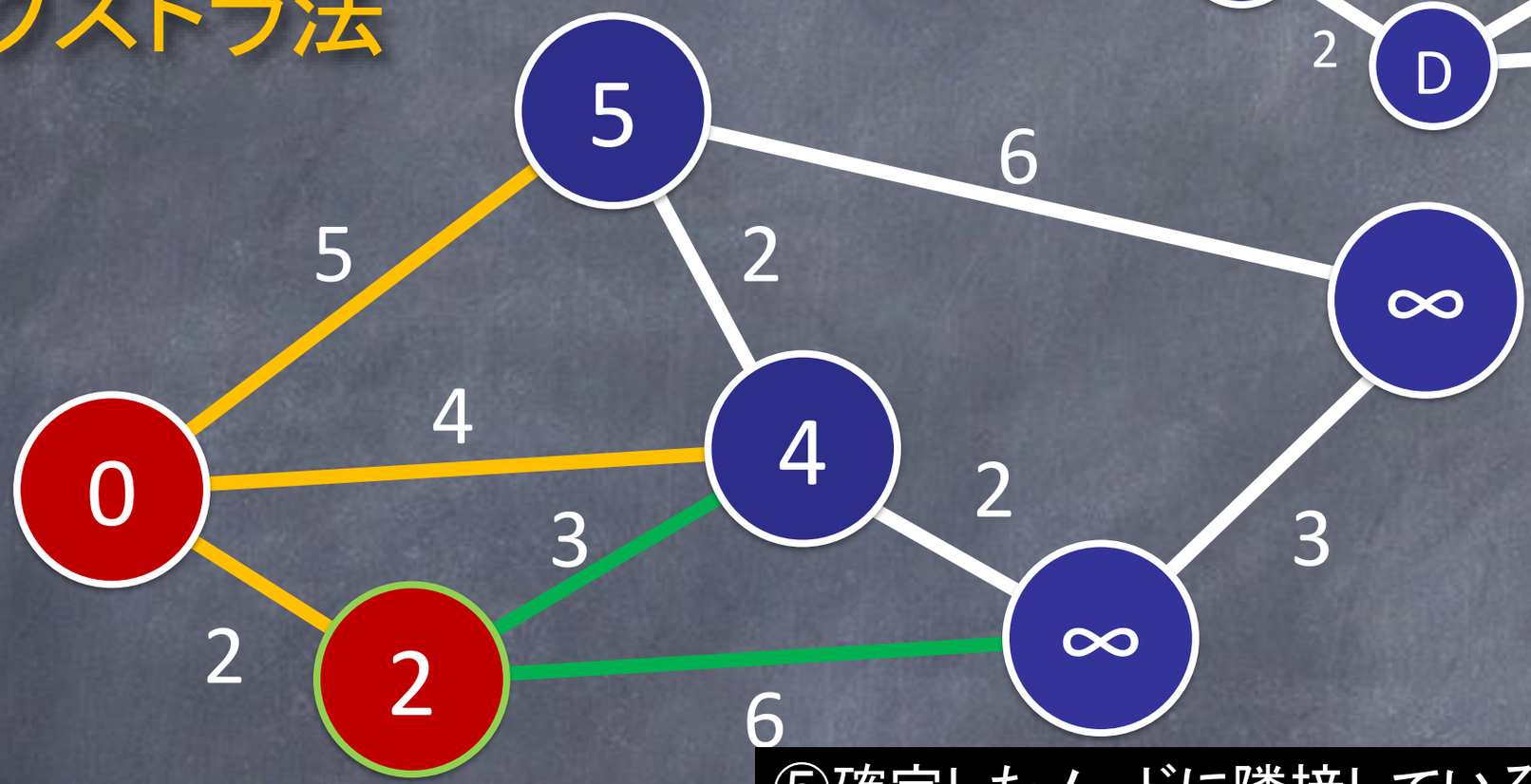
④最短距離のノードを確定する

交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	∞	
F	∞	

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法



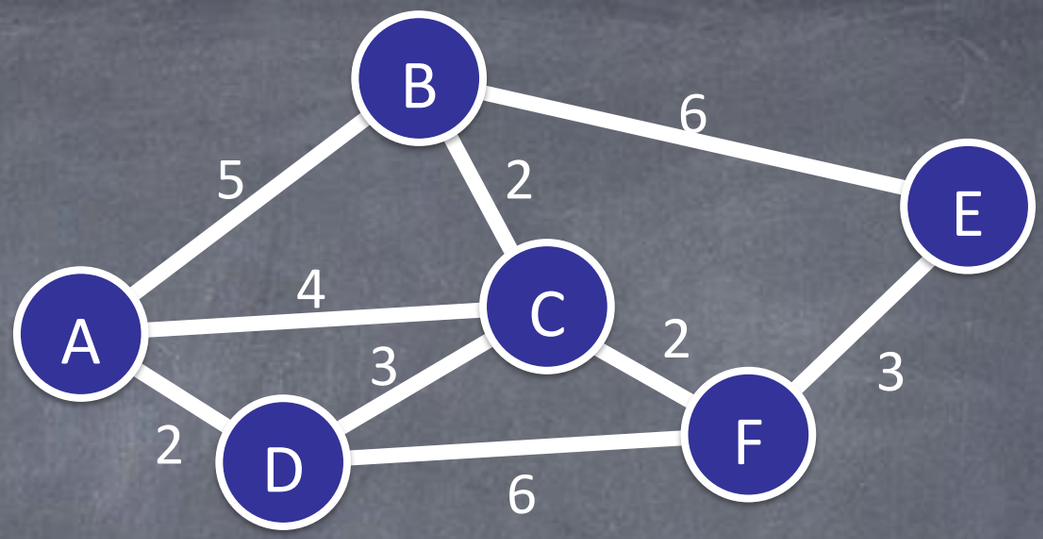


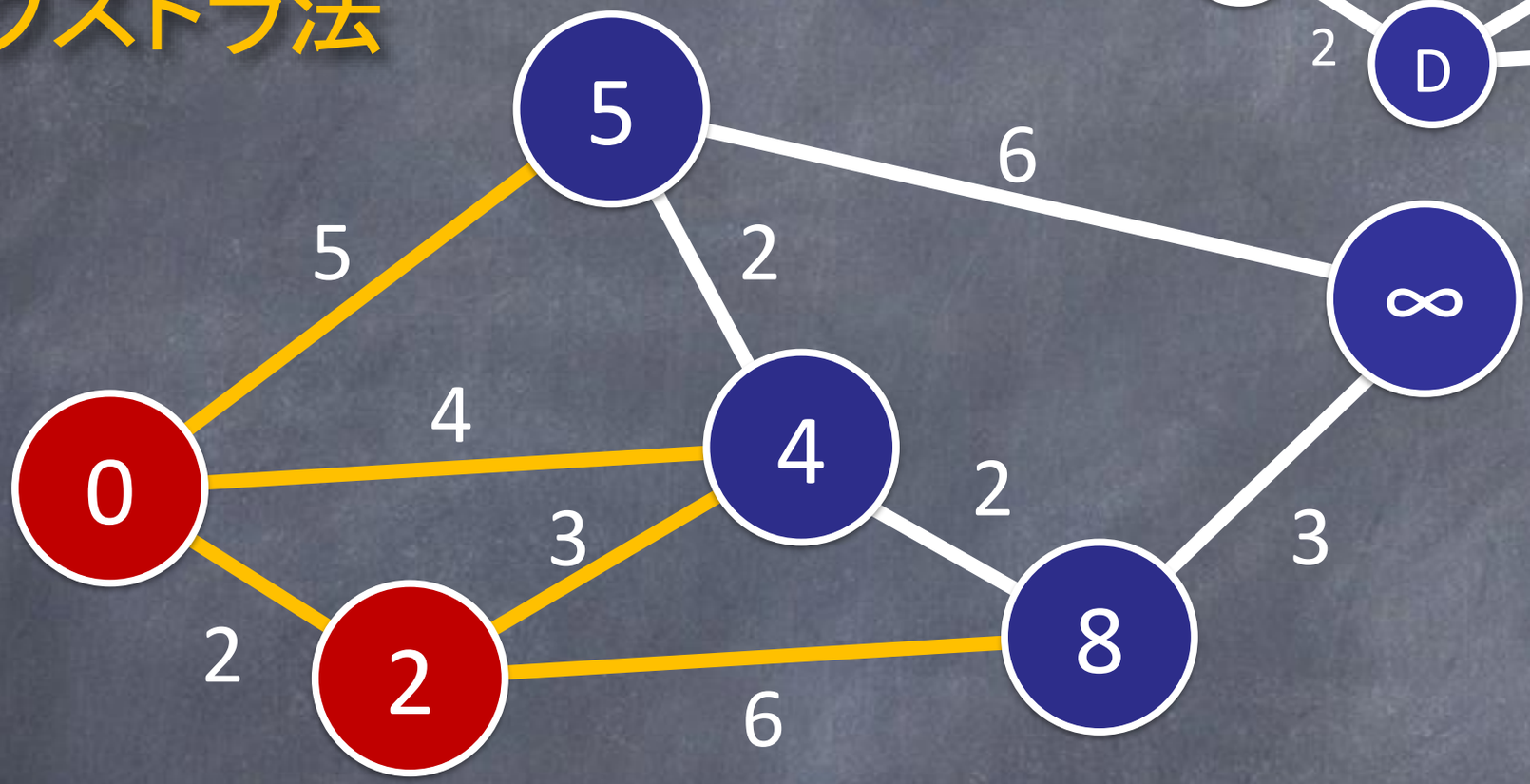
⑤確定したノードに隣接している全ノードの距離を求める

交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	∞	
F	∞	

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法



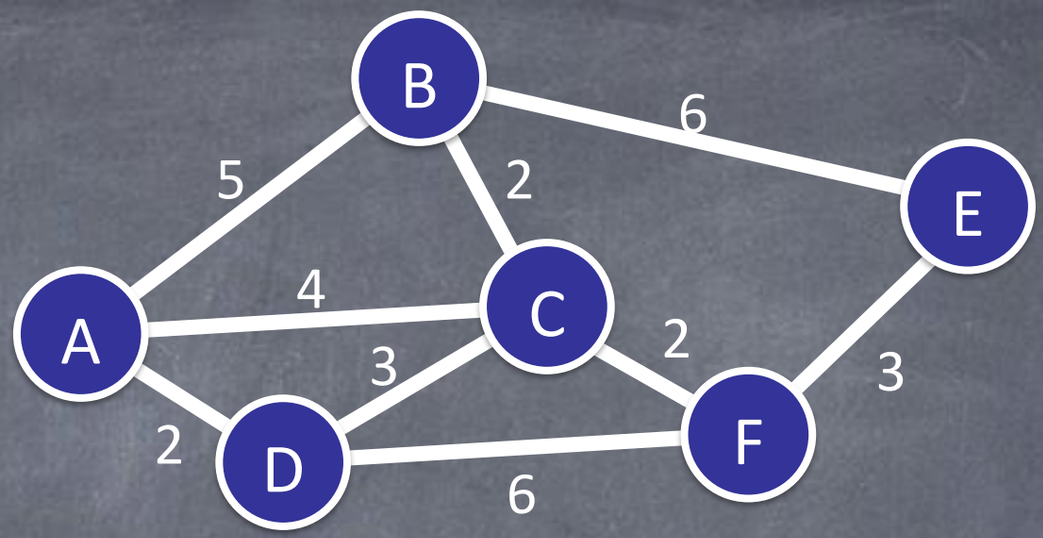


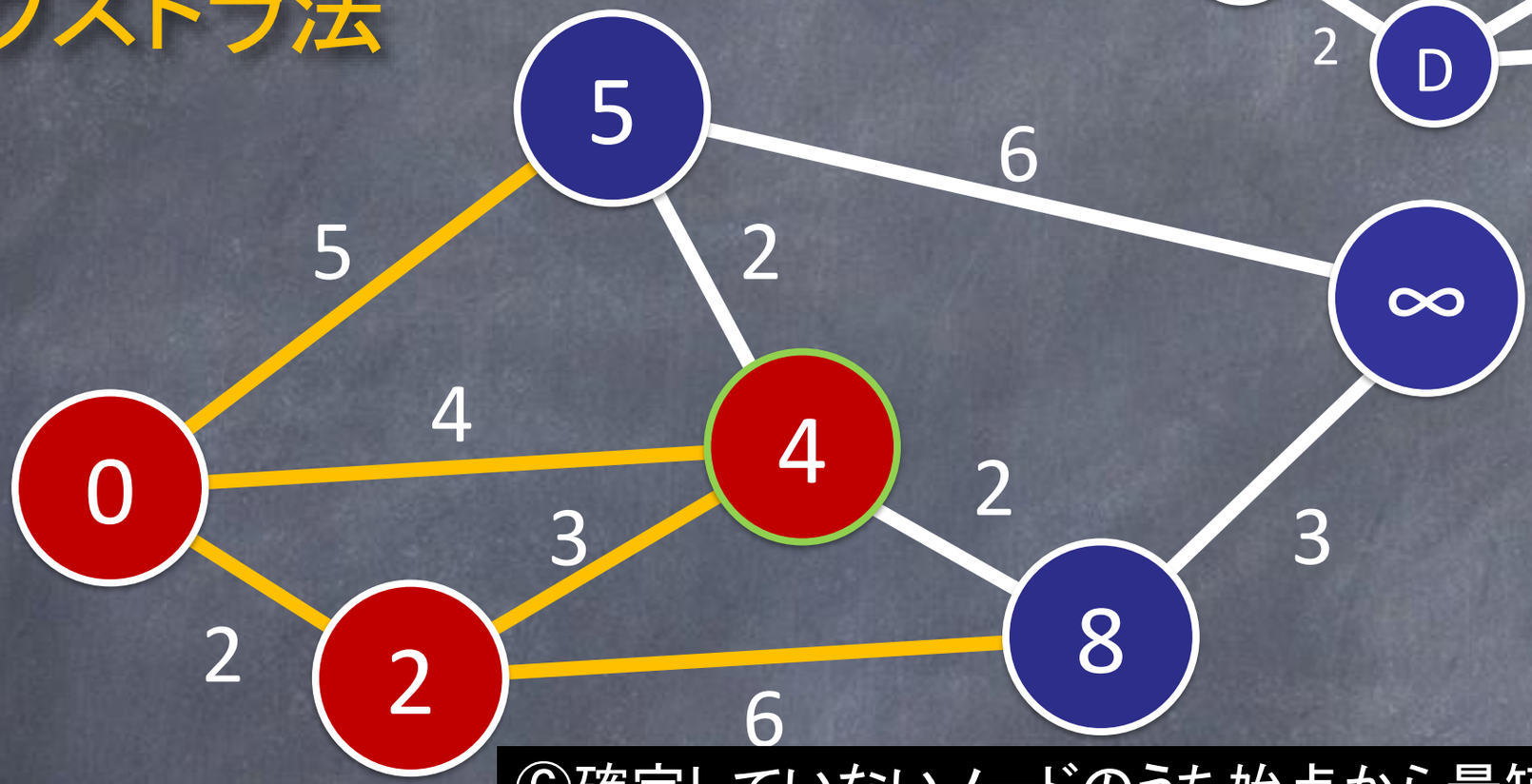
交差点	最短距離	最短経路
A	0	
B	5	
C	4	
D	2	A-D
E	∞	
F	8	A-D-F

⑤確定したノードに隣接している全ノードの距離を求める
- 新しく計算された距離が現在の距離より短い
  ⇒　新しく計算された距離を、確定距離とする
- それ以外　⇒　現在の距離を、そのまま確定距離としておく

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法



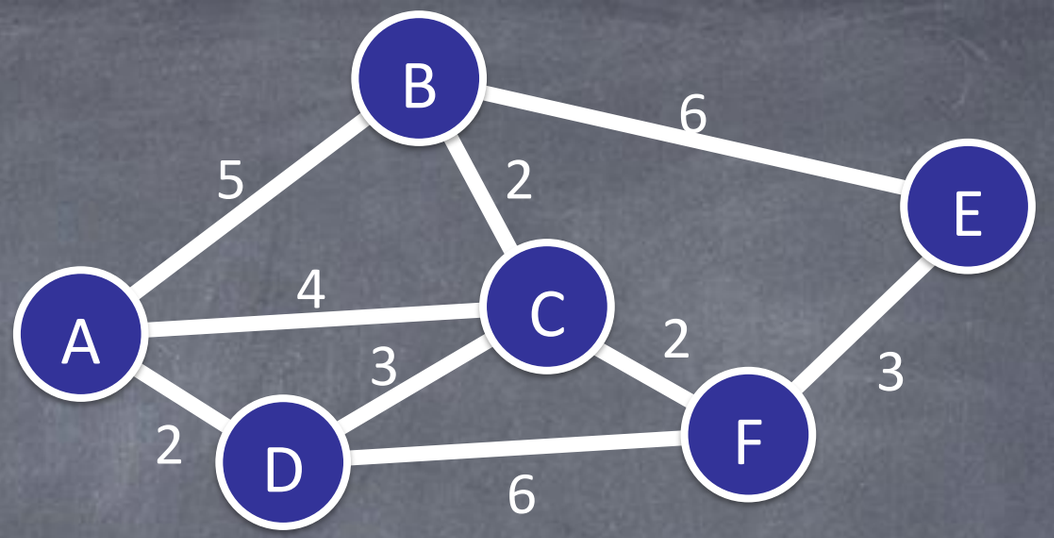


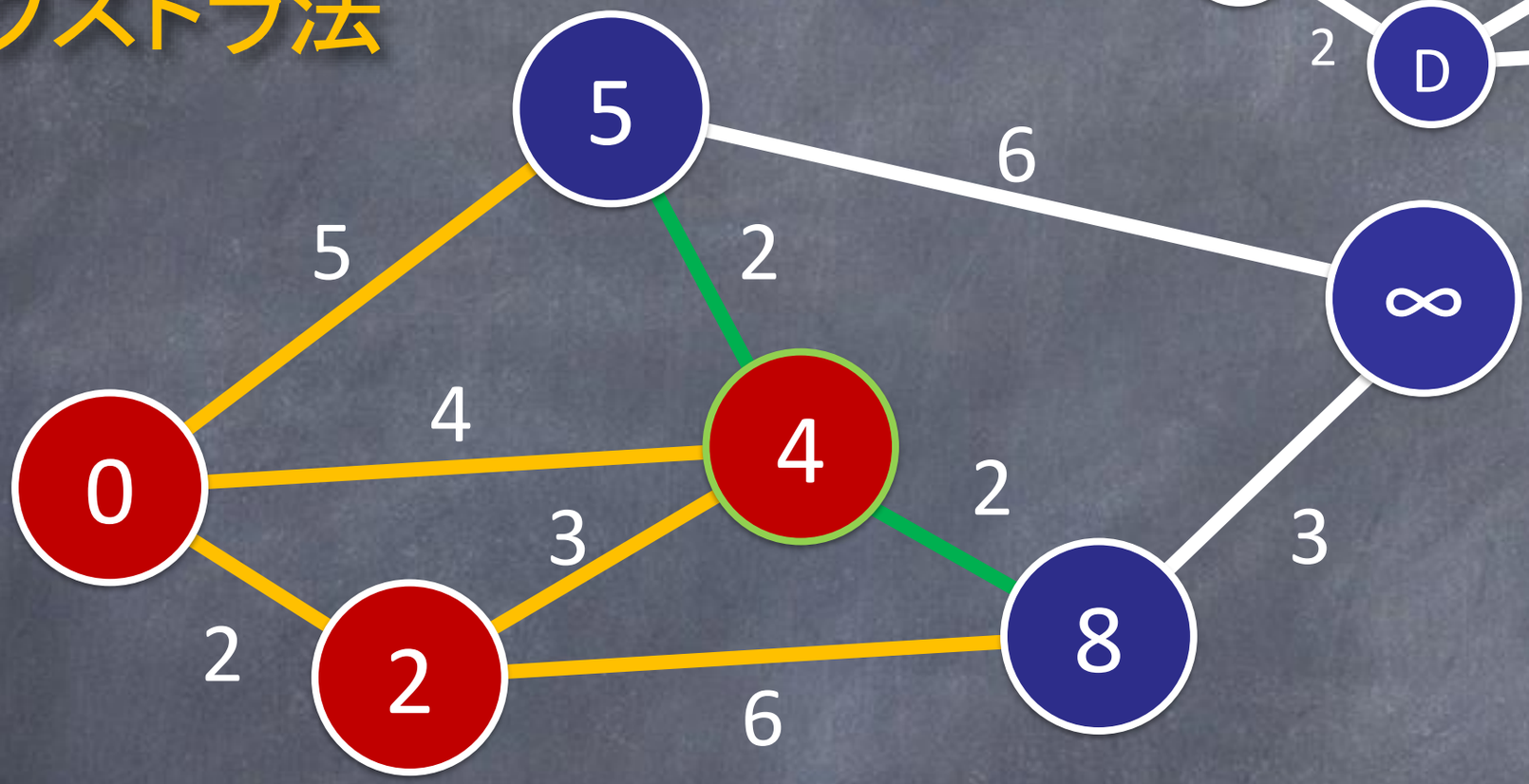
⑥確定していないノードのうち始点から最短距離のノードを確定する

交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	∞	
F	8	A-D-F

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法

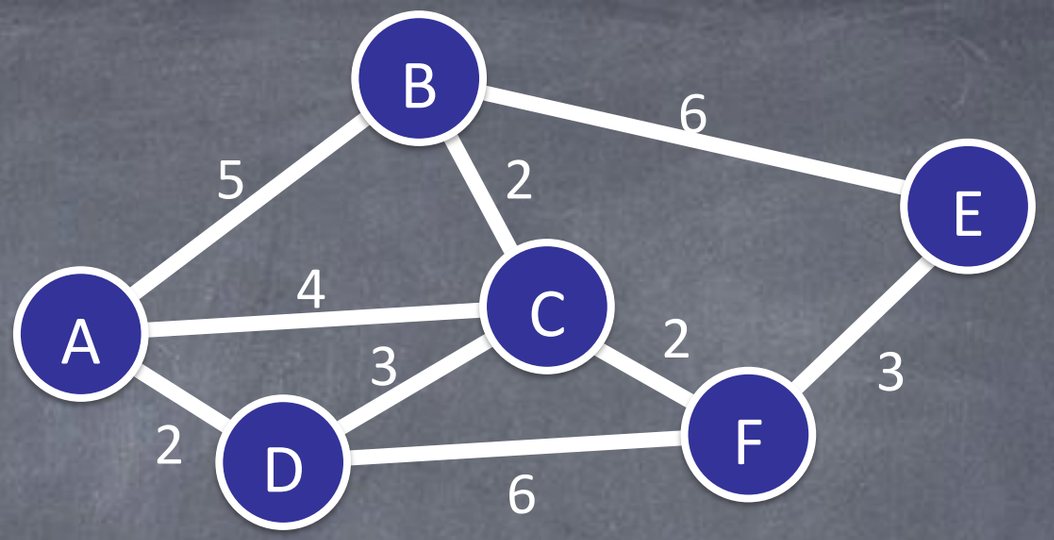


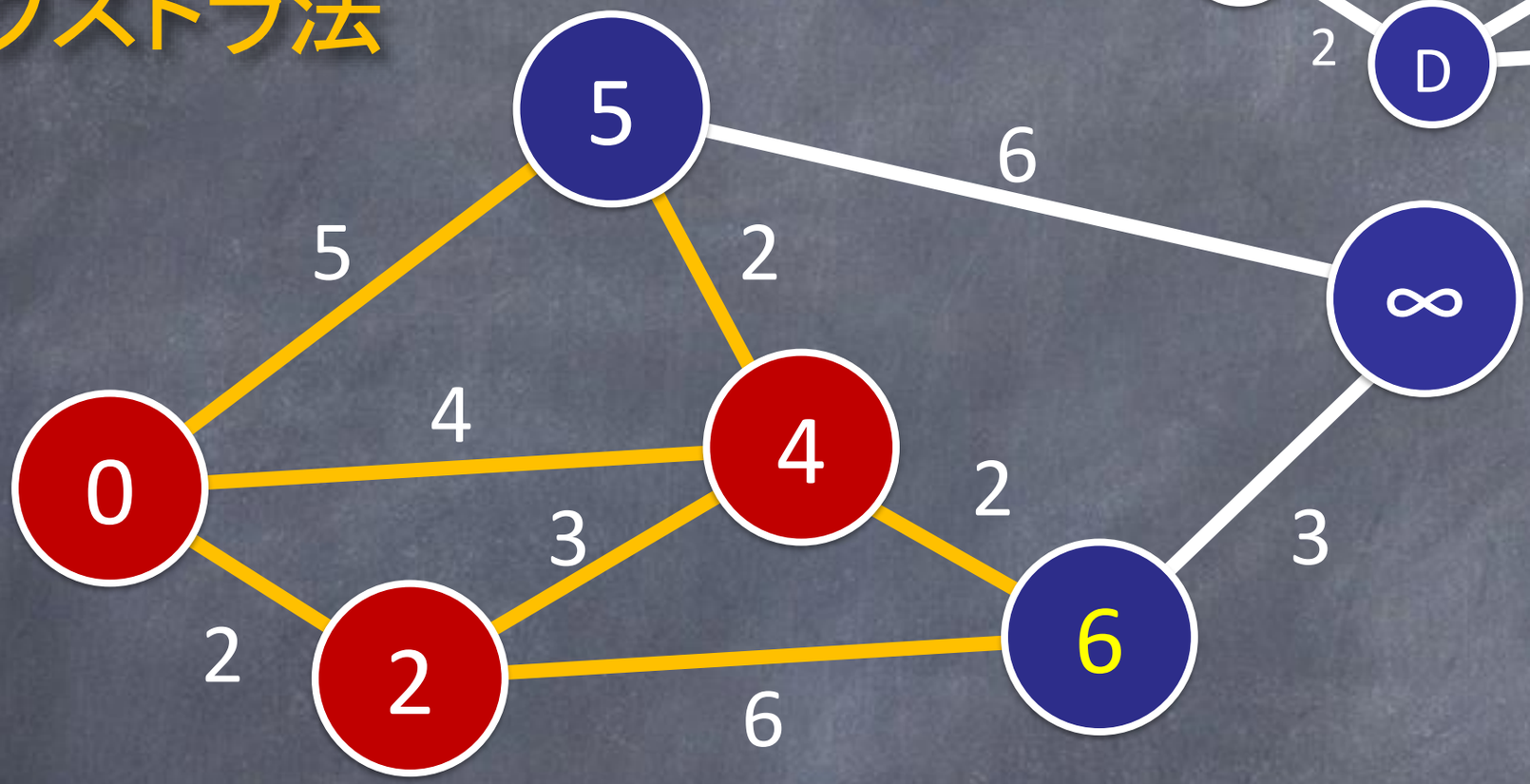


交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	∞	
F	8	A-D-F

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法

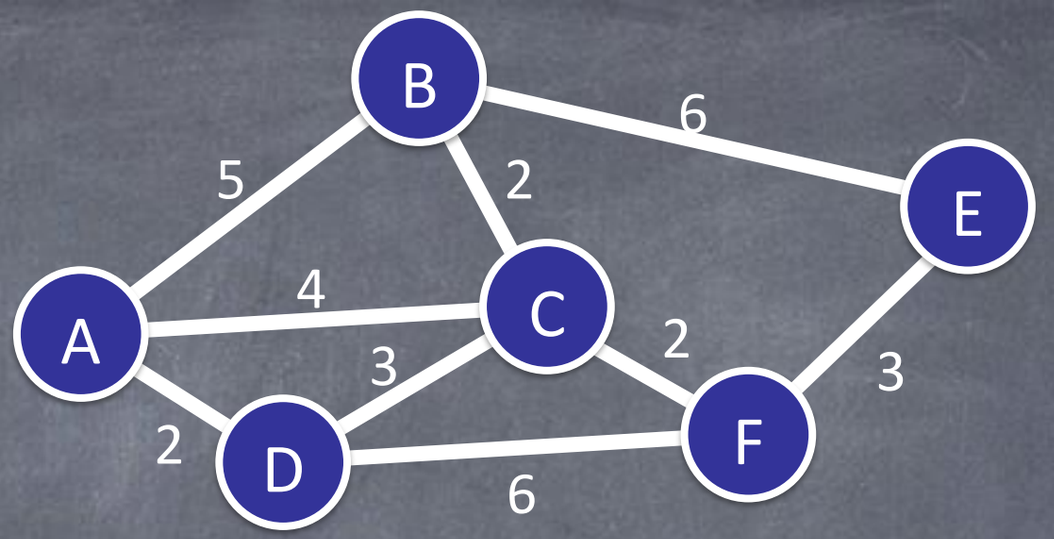


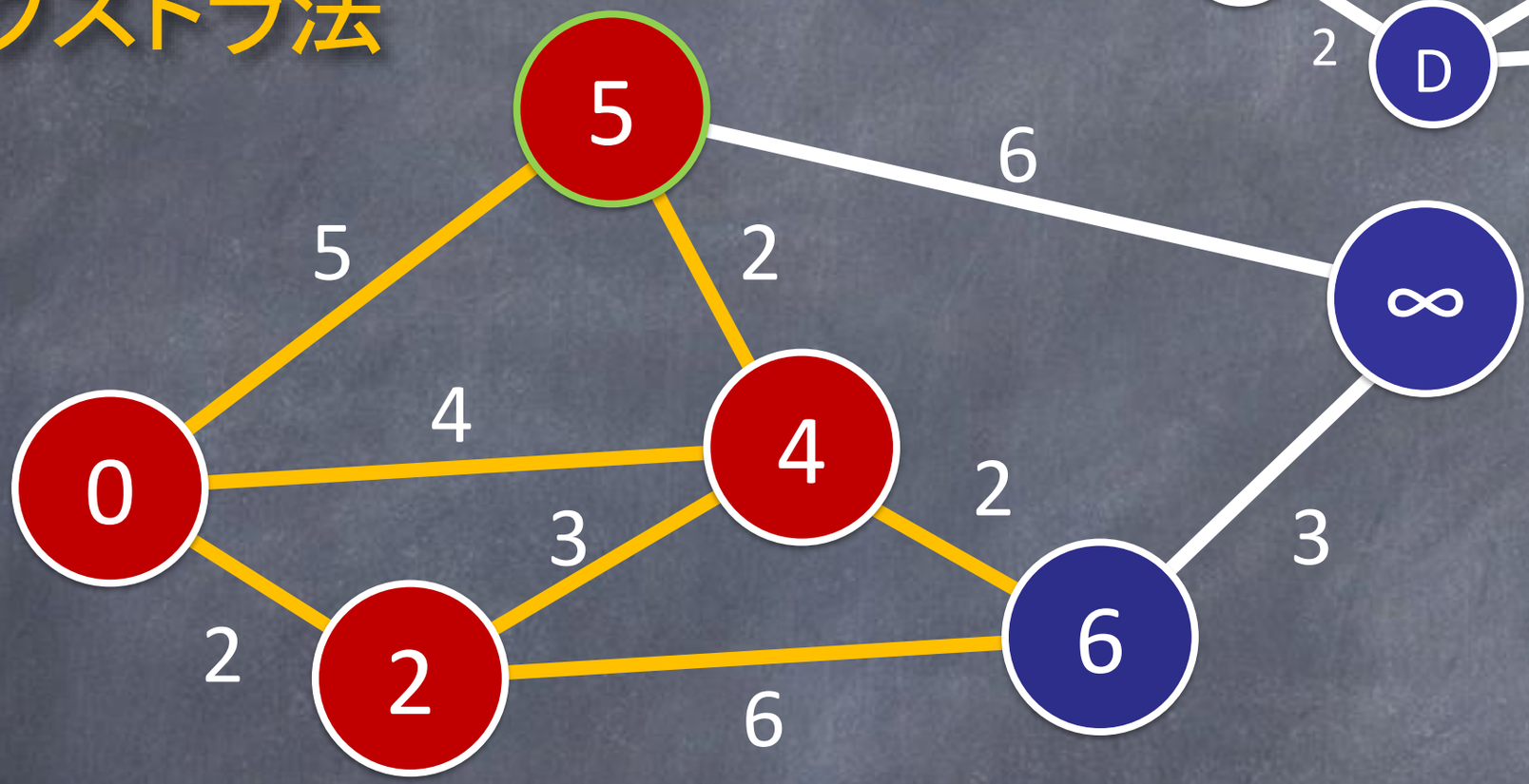


交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	∞	
F	6	A-D-F→A-C-F

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法

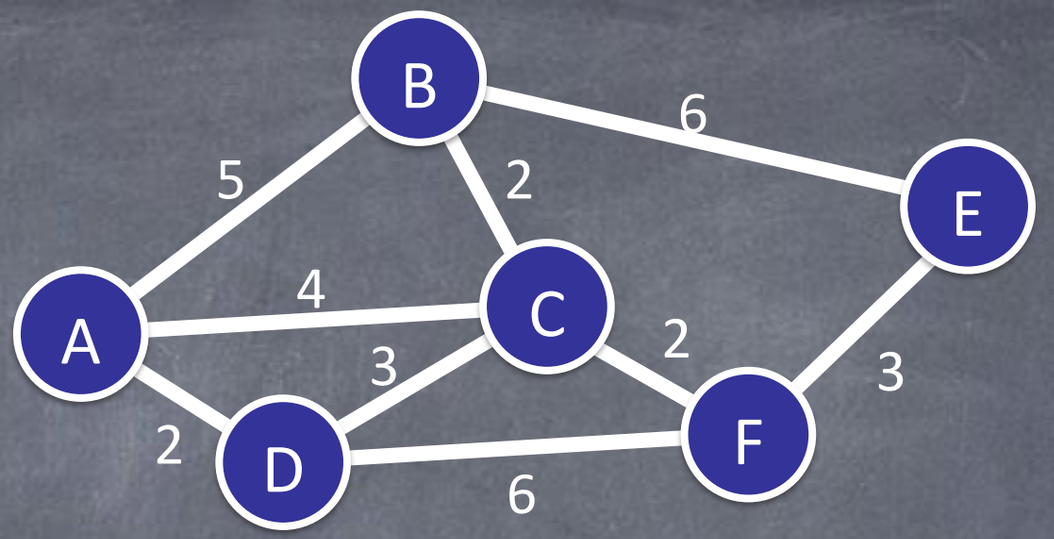


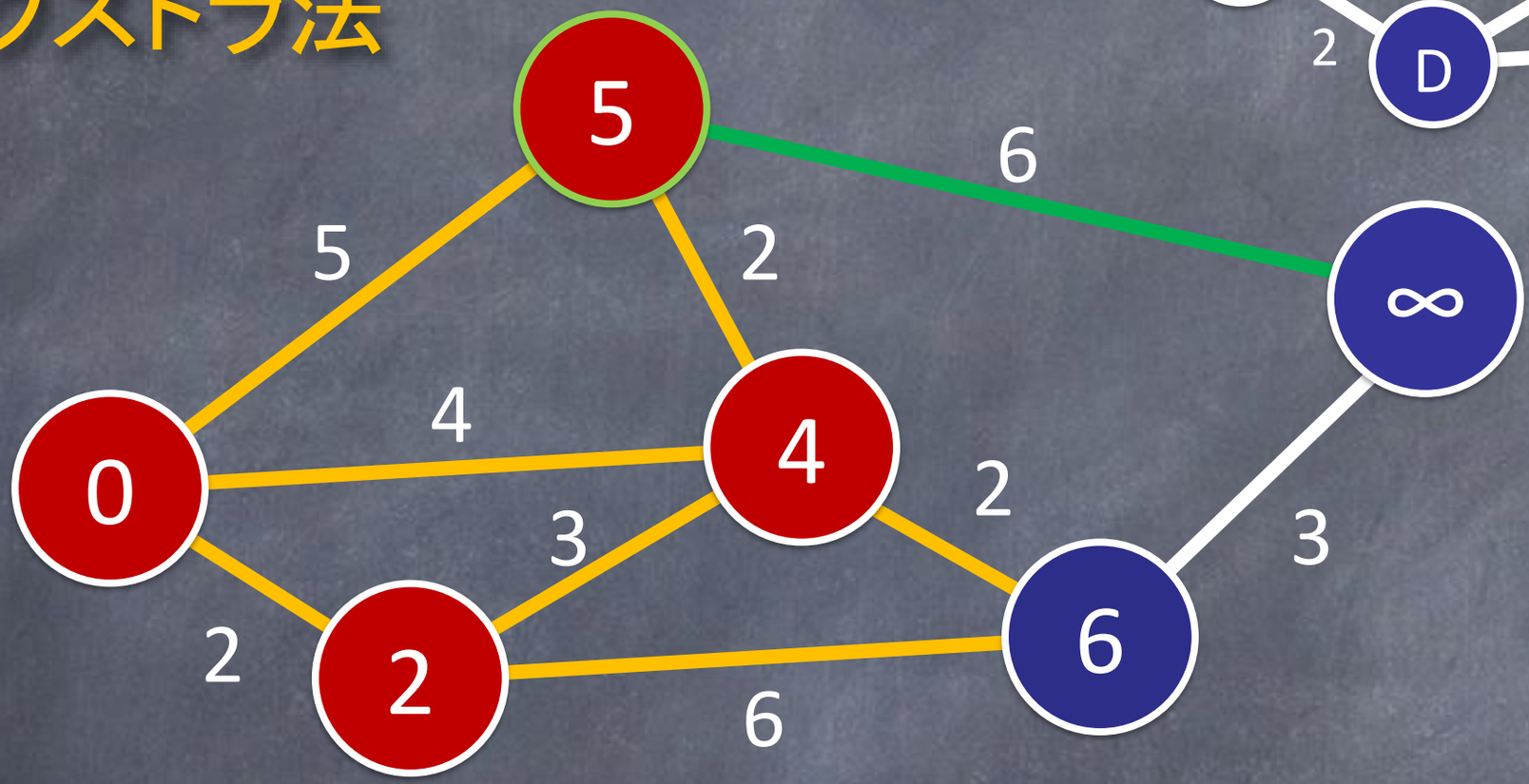


交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	∞	
F	6	A-C-F

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法

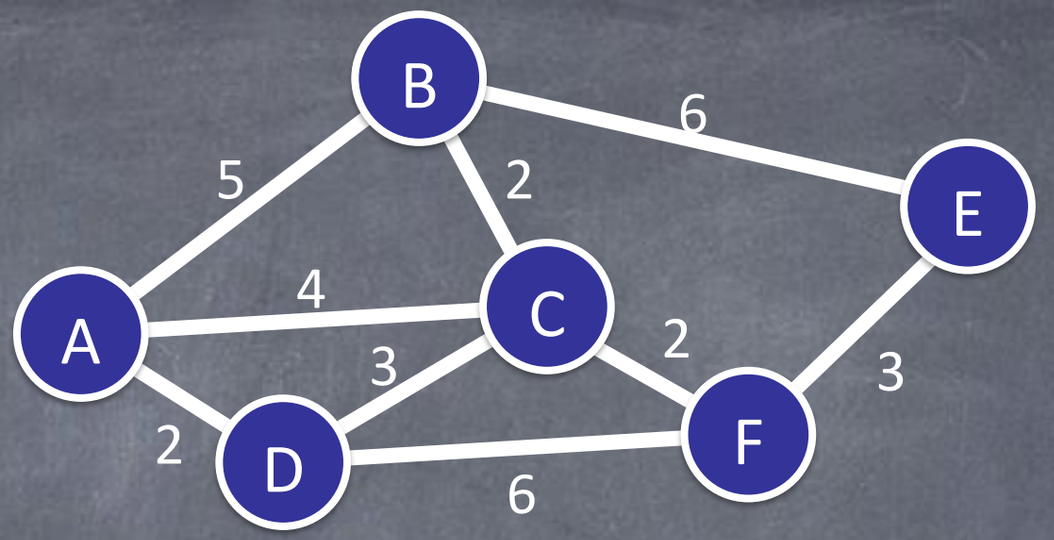


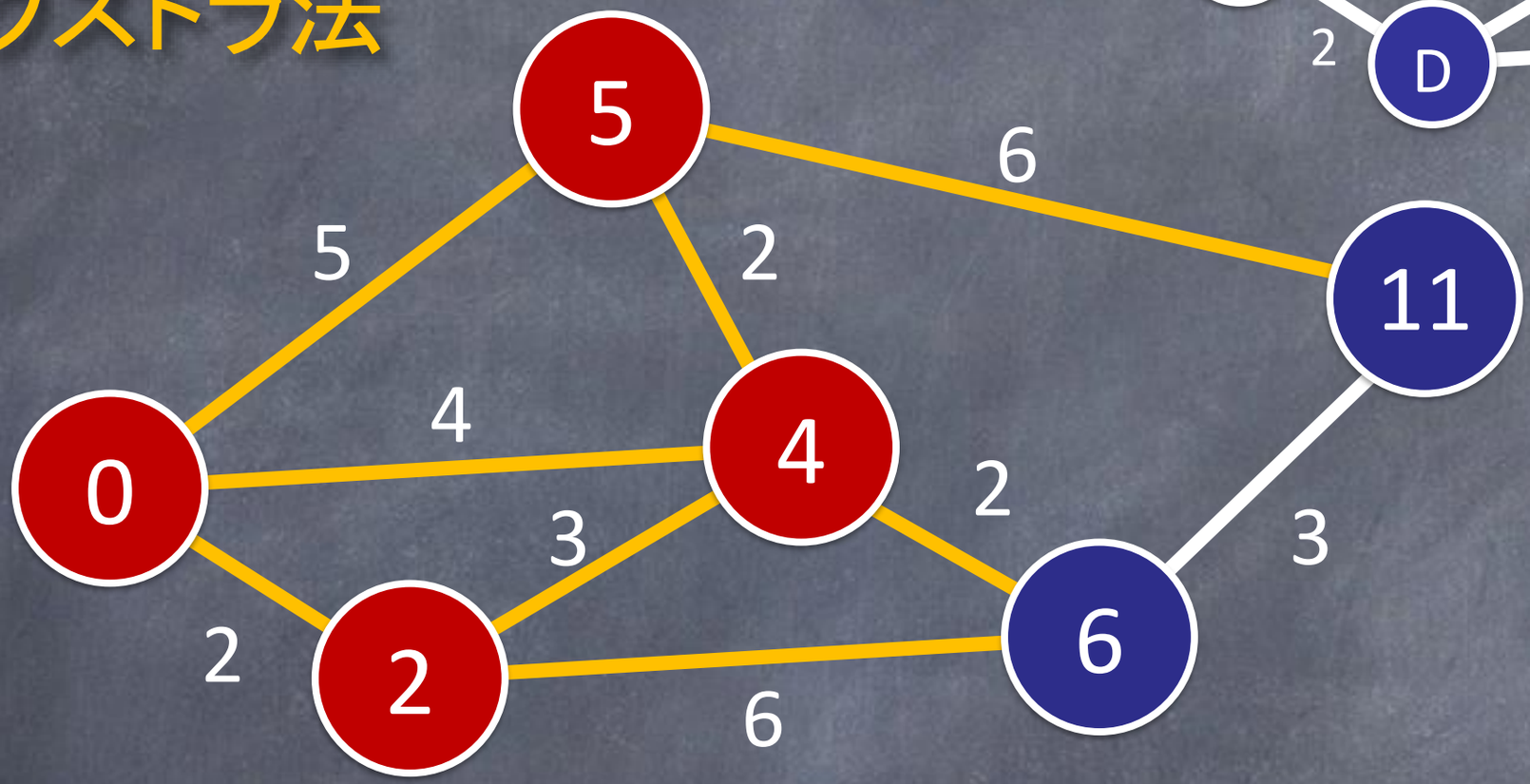


交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	∞	
F	6	A-C-F

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法



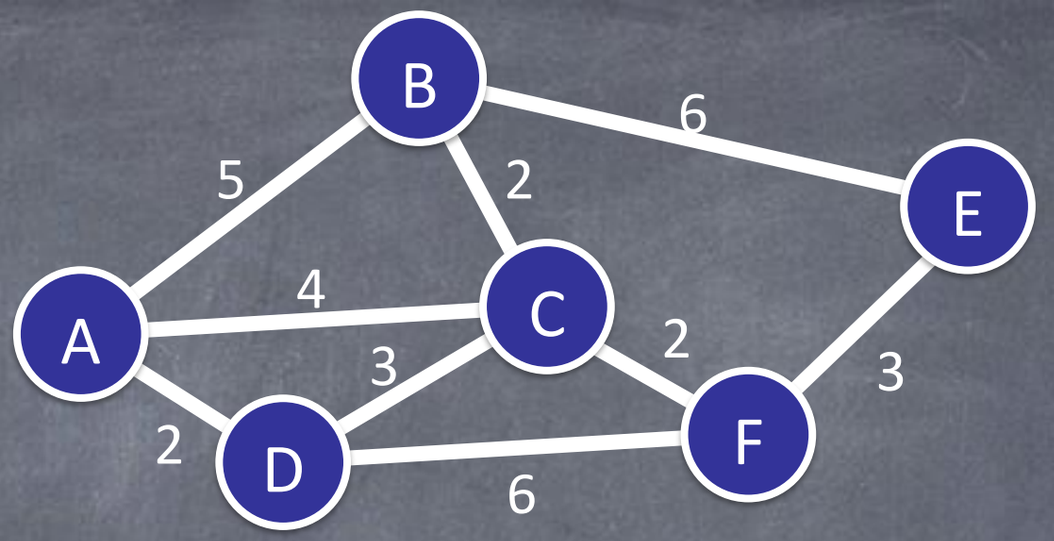


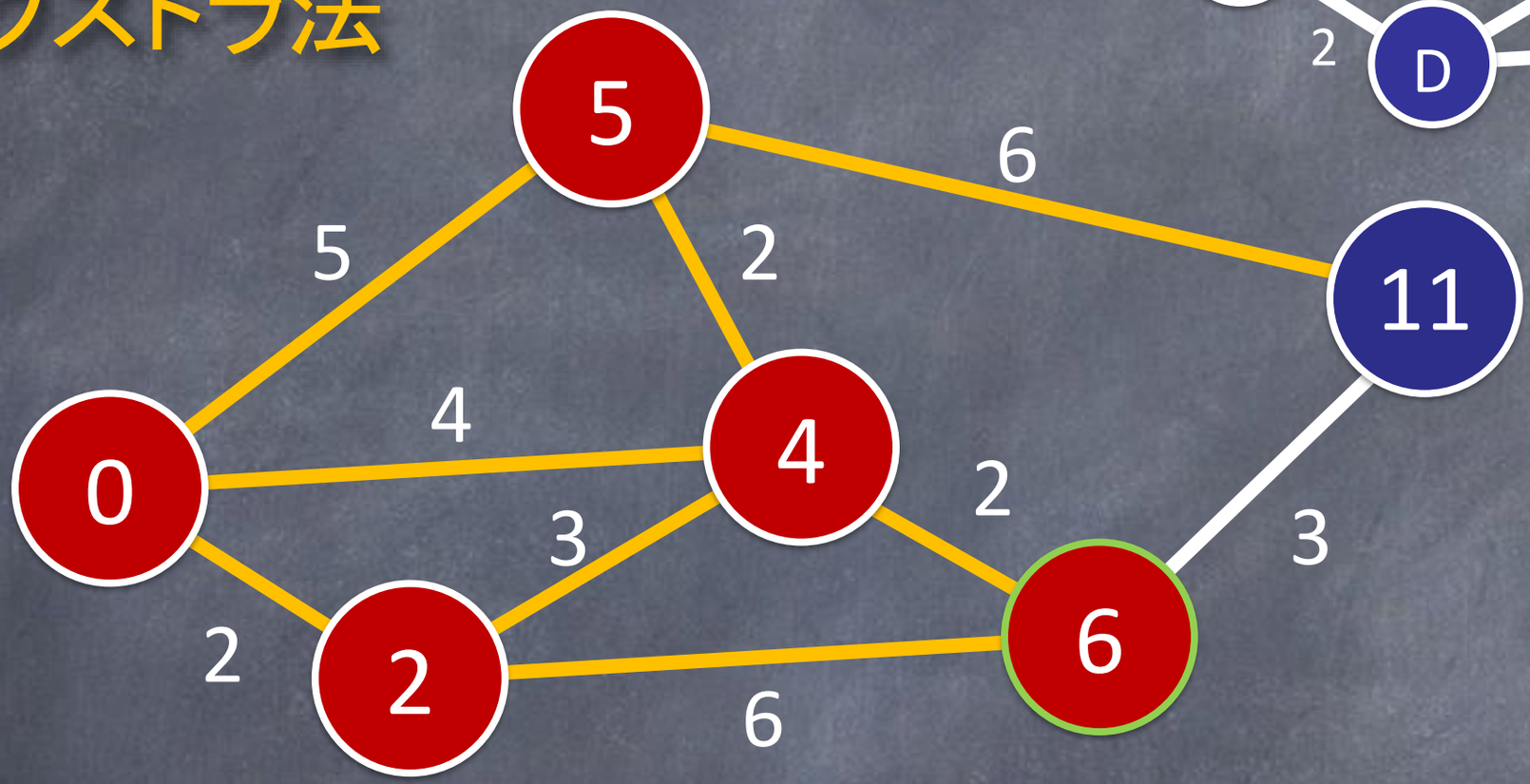
交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	11	A-B-E
F	6	A-C-F

コンピュータ実習（第10回）

最短経路探索

ダイクストラ法

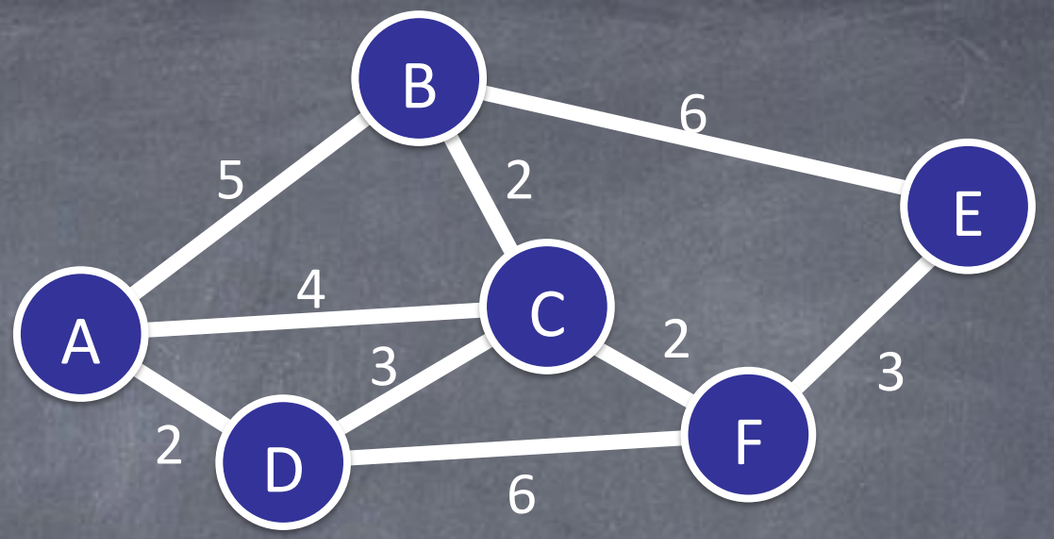


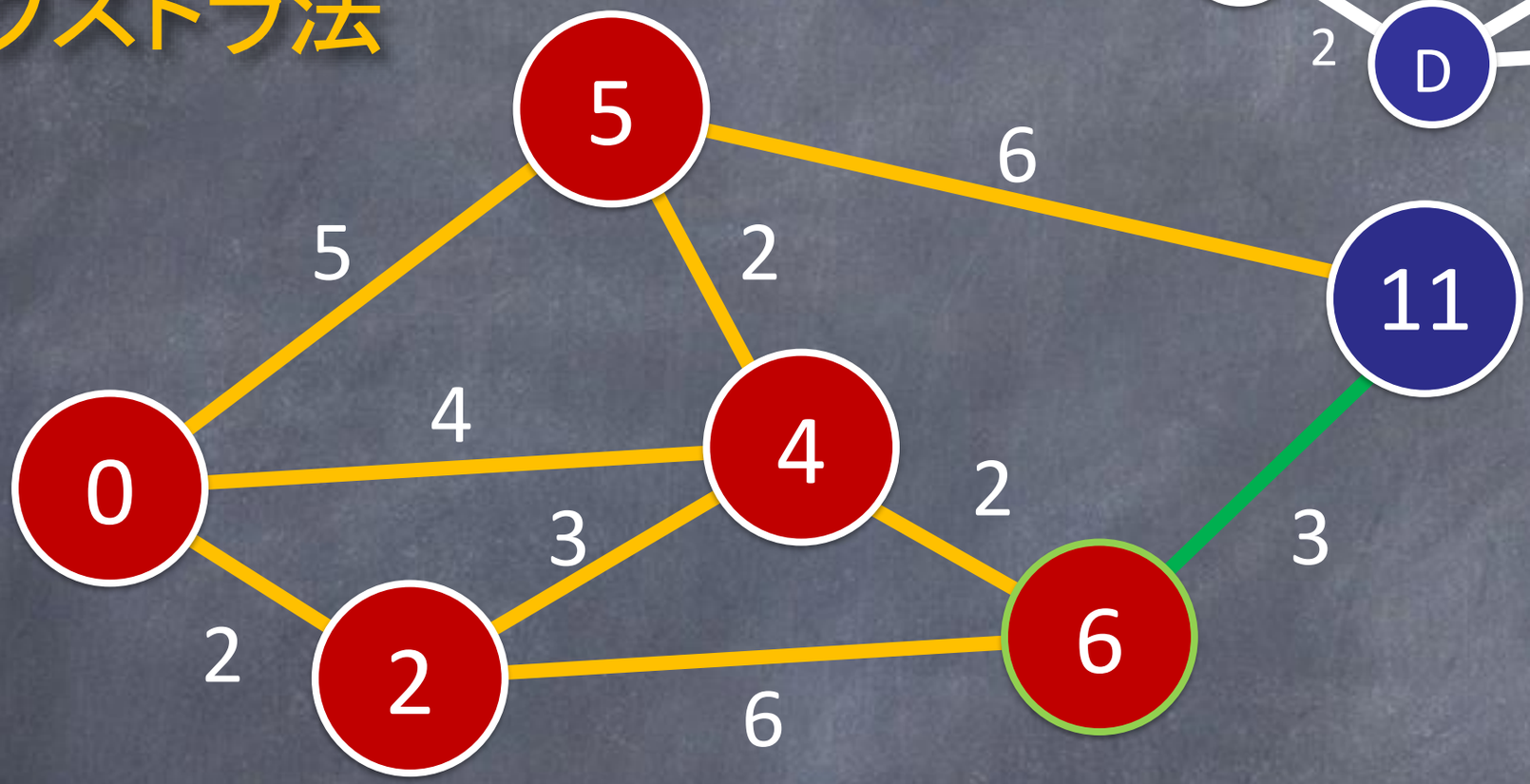


交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	11	A-B-E
F	6	A-C-F

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法

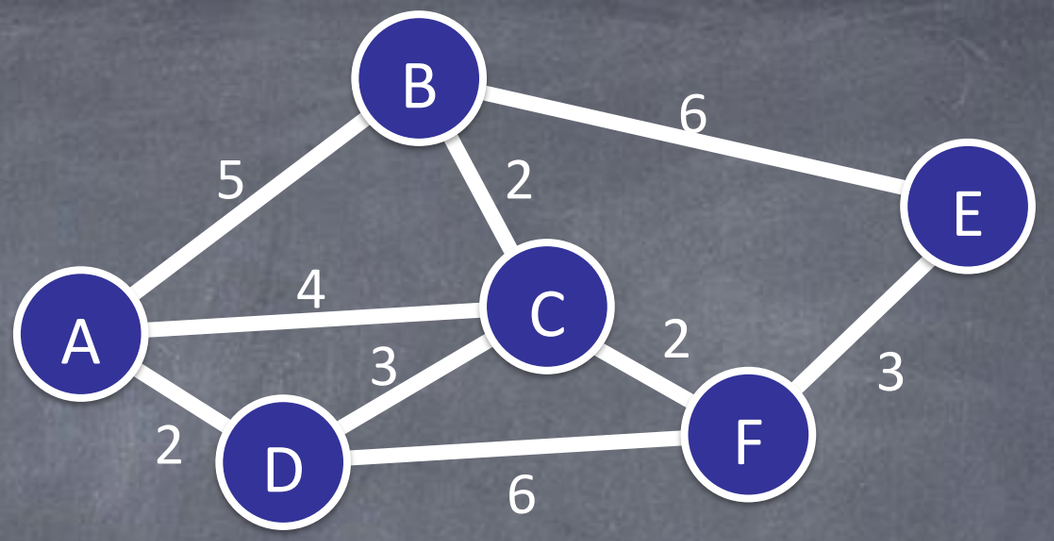


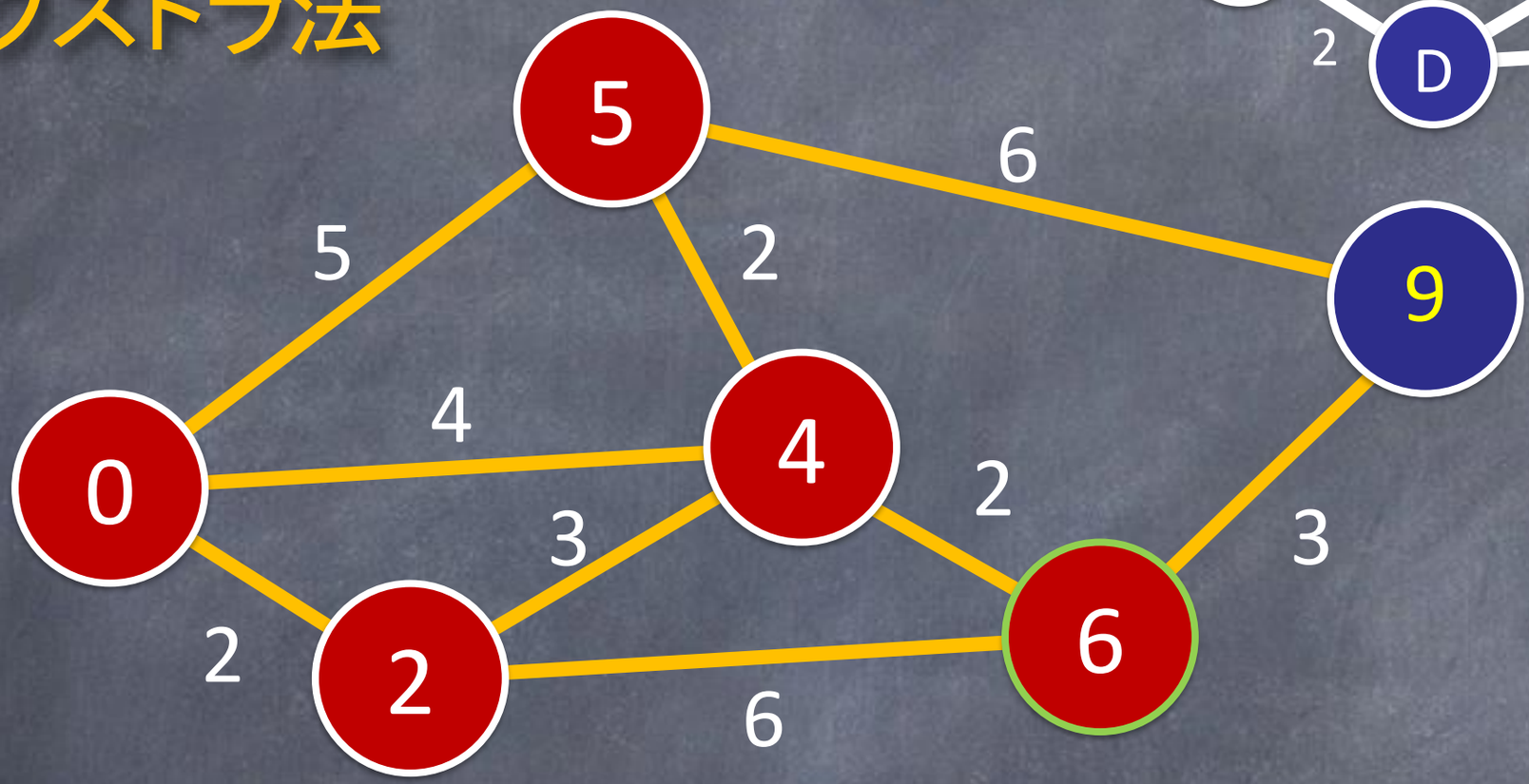


交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	11	A-B-E
F	6	A-C-F

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法

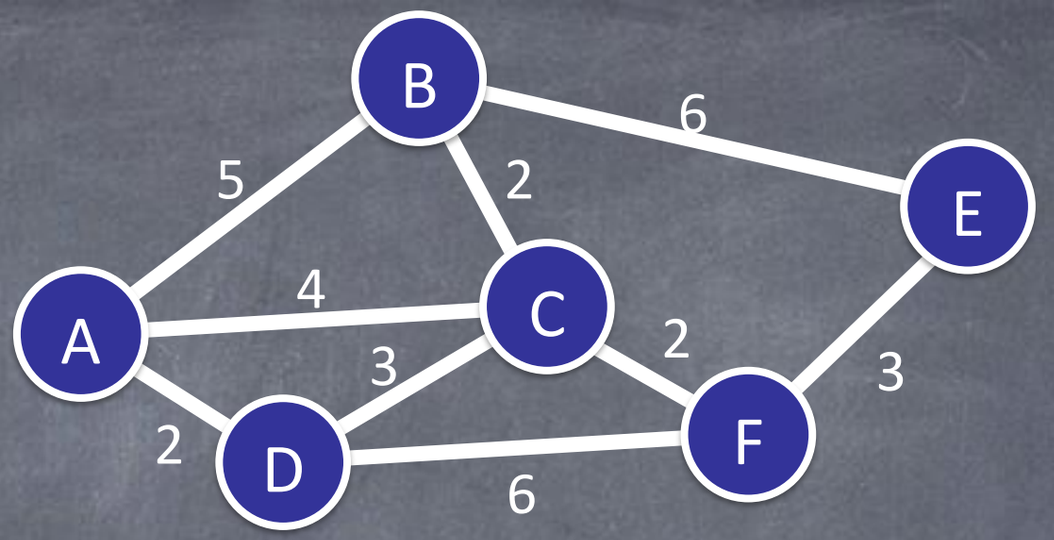


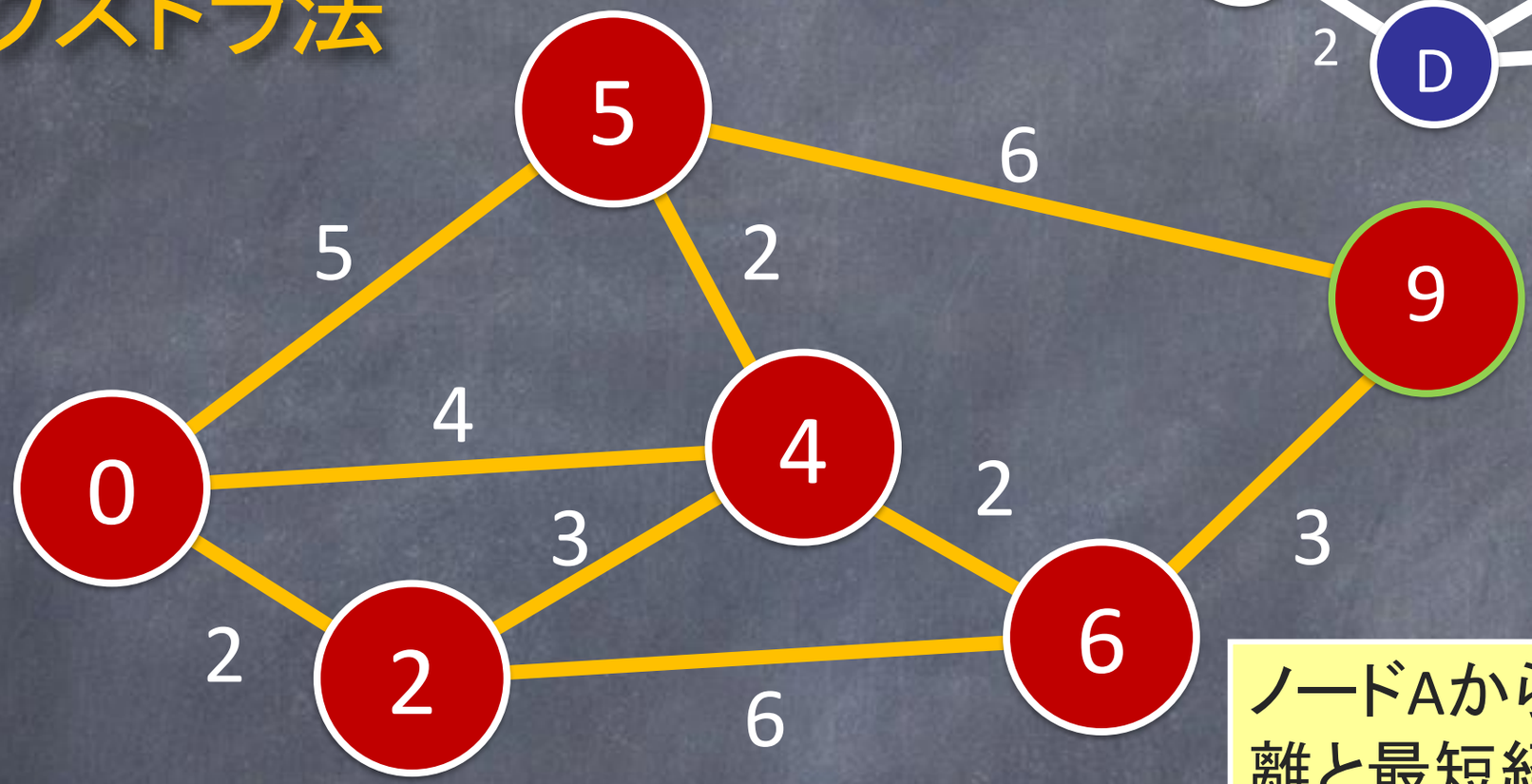


交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	9	A-B-E→A-C-F-E
F	6	A-C-F

コンピュータ実習（第10回）

最短経路探索
ダイクストラ法





ノードAから各ノードへの最短距離と最短経路が求まった

交差点	最短距離	最短経路
A	0	A
B	5	A-B
C	4	A-C
D	2	A-D
E	9	A-C-F-E
F	6	A-C-F

コンピュータ実習（第10回）

最短経路探索

ダイクストラ法のアルゴリズム（実装時）

1. 基準点（ゴール）からの最短距離が確定しているノードの集合：V
   未確定のノードの集合：U
   各ノードには，基準点からVに属するノードだけを経由してそのノードまでに達成する最短経路の長さdistanceを与えておく．
   最初は，Vに属するノードは基準点しかない．distance=0
   Uに属するノードは基準点からの最短経路がまだ見つかっていないから，そのノードのdistanceには，∞（あり得ない大きい値）を与える．
2. 未確定のノードから基準点までの距離が最小のノードを新たに確定する．
3. 確定したノードに隣接するすべてのノードに対して，確定であれば何もしない．未確定であれば，確定したノードからの距離と確定したノードのdistanceの和を算出し，隣接ノードのdistanceより小さい場合，隣接ノードのdistanceを更新する．
4. すべてのノードが確定したら終わる．そうでなければ，2に戻る．

コンピュータ実習（第10回）

最短経路探索

プログラムのポイント

- 集合U（最短距離が確定していないノードの集合）と集合V（最短距離が確定しているノードの集合）を区別する方法

- 集合Uの中で，distanceが定まらない交差点を除外する方法

- distanceの算定方法

コンピュータ実習（第10回）

最短経路探索

プログラムのポイント

- 集合Uと集合Vを区別する
  - 俗に「フラグ」と呼ばれる変数を用意する.

- フラグというのは，ある状態が満たされたかどうかを記憶するための変数を指す言葉で，大抵はゼロとゼロ以外の数字（1など）が代入されます.
  - 最初は各交差点のフラグをクリアしておき（集合Uに分類），
  - 確定するときにフラグをセットする（集合Vに分類）.

- フラグの中身はif文で判定して処理を選択します.

コンピュータ実習（第10回）

最短経路探索

プログラムのポイント

- 集合Uの中で，distanceが定まらない交差点の扱い
  - distanceに十分大きな値を代入する.
    - distanceを決定した後で「distanceが最小のものを選ぶ」ため，distanceを大きな値にしておけば選ばれる心配がないため，除外したのと等価になります.
- 最短になる方向に処理が進むので，値が大きくなることはないため，最初に大きな値を代入しておいて，交差点を一個確定する毎に隣接する交差点のdistanceを更新するだけで，「Vを経由した最短経路の長さ」が算出されることになる.
- 例外的に，基準交差点（最初の交差点）の距離だけゼロにします.
  - 物理的には当たり前ですが，プログラムとしてもこれによって最初にこの交差点が確定され，それに隣接する交差点のdistanceが計算されることになります.

コンピュータ実習（第10回）

最短経路探索

プログラムの考え方

1. 変数の準備
   - 交差点のdistance
     - これはあとで使うのでCrossing構造体の中に用意する
   - 交差点のフラグ
     - これはダイクストラ法の計算が終わったら不要になるので，関数内で読み使用する配列変数にする．

2. 変数の初期化
   - 各交差点のdistanceを十分大きな値（例えば，1e100=1*10^100）に初期化し，基準交差点の数値のみゼロにする．
   - 全交差点のフラグ変数を全部クリアする（ゼロを代入することで所属を集合Uに設定）

コンピュータ実習（第10回）

最短経路探索

プログラムの考え方

3. 繰り返し処理

- 全交差点のフラグを調べつつ，まだ最短距離が確定していない交差点の中から，distanceが最小のものを選び出す．
- この交差点のフラグを立て，確定扱いにする（1を代入することで所属を集合Vに設定）
- この交差点に隣接する各交差点において，今確定した交差点のもつdistanceと，その隣接する交差点間の距離の和が，隣接交差点の持つdistanceより小さい場合，隣接交差点のdistanceに前者を代入
- 以上の処理を未確定の交差点がなくなるまで繰り返す．このことは，この処理ループを交差点数だけ繰り返せば終わることを意味する．

4. 処理結果

- この時点で，すべての交差点のゴールまでの最短距離が計算されている．

コンピュータ実習（第10回）

ダイクストラ法による最短経路探索
今日の演習

- 演習１
  - 適当に決めたゴールまでの各交差点からの最短距離をダイクストラ法により求めるプログラムを作ります.
    最短距離変数はあとで使うので，構造体Crossingにdouble distance;として追加し，フラグはダイクストラ法を実装した関数内でint done[CrossingNumber];として宣言する.
    結果（最短距離を含む）は適当に画面に表示すること
- 演習２
  - 上のプログラムに最短経路を導出するプログラムを追加し，適当に設定したスタート地点からゴールまでの最短経路を 画面に表示するプログラムを作ります.
    （グラフィックで表示 する必要はありません．）

コンピュータ実習（第10回）

ダイクストラ法による最短経路探索
演習2のヒント

- 演習１のプログラムで交差点情報の構造体に追加された、int previousには、ダイクストラ法による各交差点の最短距離の導出過程で、目的地から数えて直前の交差点番号が入力されている。（すなわち、出発地のpreviousには、最短経路で次に進む交差点番号が入力されている。）
出発地から順番にpreviousの数値を拾ってpathに与えれば、出発地から目的地までのpathが取得できる。
最後に経路の終了フラグ path[i]=-1 を書き込めば終わり。