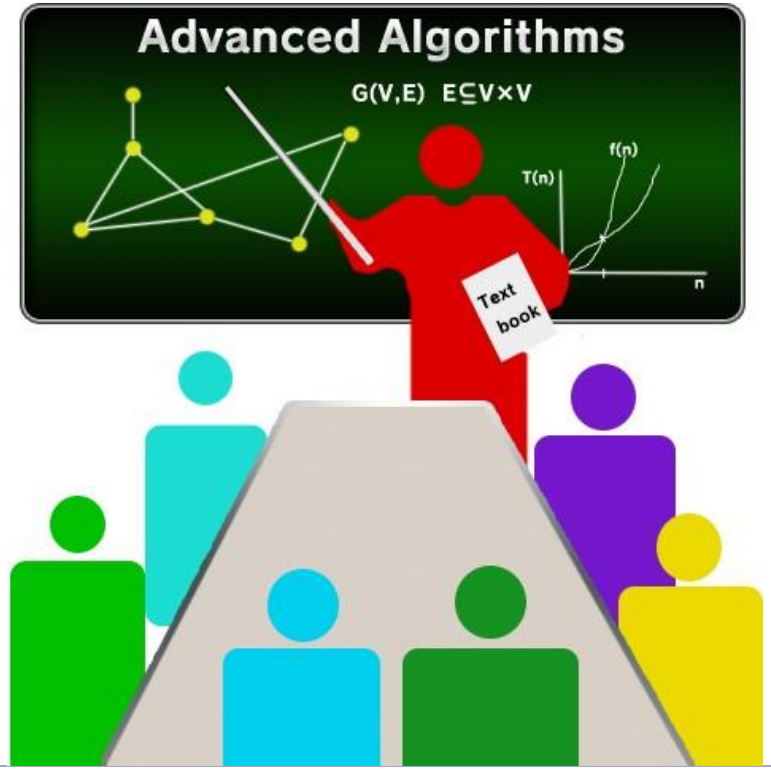

アルゴリズム特論 [AA201X]
Advanced Algorithms

Lecture05. Shortest Path Problem

Exercise 05 のために

重み付きグラフ（2）

- **最短経路木の構成方法**
- **Dijkstra'sAlgorithm**

⇒**前回(Ex04)の** Prim'sAlgorithm **との類似点・相違点**

Prim's Algorithm を理解していますか？

Dijkstra's Algorithm

- グラフG(V,E,W) 上で探索
- 与えられた頂点（始点）から他の各頂点へ至るための最小コストを計算する

- DijkstraʼsAlgorithm で、与えられた始点から他の全ての頂点への最短距離を探し終わると、グラフG上に全域木ができあがっている　⇒　最短経路木

- ただし、各辺の重み $w \in W$ は $w > 0$ でなければならない（DijkstraʼsAlgorithm が正しく動作するための前提 ⇒ もし $w < 0$ があったら？…演習課題で）

Dijkstra's Algorithm

- **Primと同じところ**
- 任意の始点 s が与えられている

⇒ 部分解 T = {s} が 徐々にグラフG全体を埋め尽くしていくイメージ

- T に含まれる頂点から出る辺を辿って、V -T に含まれる頂点を探す
- V =T になるまで処理を繰り返す

- **Prim と違うところ**
- T からV-T への距離のアップデート方法が違う

⇒ Prim の場合：T からV-Tへの現在の最短距離 と 新しくT に追加されるV-Tの頂点から他のV-Tの頂点へ向かう最短距離 を比較

⇒ Dijkstra の場合 ： T からV-Tへの現在の最短距離 とTから 新しくT に追加されるV-Tの頂点への距離 ＋ そのV-T の頂点から他のV-Tの頂点へ向かう最短距離 を比較

Dijkstra's Algorithm

- 前処理（初期化）

```
for ( i = 0; i < n; i++ ) {
    label[i] = 0;
    d[i] = D[root][i];
}
adj[root] = -1;
label[root] = ++visited_order;
```

- ルーチン本体

```
/* while (S != V) { */
while (visited_order < n) { /* S !=V  -->  |V| < |S| */
  min_cost = INFTY;
   for ( i = 0; i < n; i++ ){ // Choose a vertex w in V
     if(label[i] != 0) continue;                    // -S

    //such that...
     if ( /* [ Complete Here!! ] */  ){// d[w] is a minimum;
         min_cost = /* [ Complete Here!! ] */ ;
         min_node = /* [ Complete Here!! ] */ ;
           }
   }
     if ( min_cost == INFTY ) continue; /* no edges from v to w */
     // add w to S;
   label[min_node]= ++visited_order;
```

- V から V-T への最短距離を調べる（＋V へ追加）
- VからV-Tへの最短距離を更新する

if 文 の条件が Primと異なる）

```
   for ( i = 0; i < n; i++ ){    //for (v in V
  if(label[i] != 0) continue;           // -S)

  //d[v] = min{d[v],d[w]+D[w,v]};
        if ( /* [ Complete Here!! ] */ ){
     d[i] = /* [ Complete Here!! ] */ ;
     adj[i] = min_node;
       }

   }
}
```

Tips

- 青い部分（最短距離検索）は、線形探索 O(|V|) になっているが、もちろんヒープ化すると 頂点数が非常に多い場合での高速化に繋がる

- もし全ての辺の重みが 1 だったら…？

⇒Dijkstra で探すことは可能だが、

素直に 重みを考慮しない DFS や BFS を利用したほうが処理が早い