まとめ
Summary

2019年度

担当：只木進一（理工学部）

数学的背景

- 集合とその演算
  - 集合の演算記号を理解していること
    - 集合の差：$A \setminus B$
    - べき集合：2^A
- 数学的帰納法
  - 初期値(例えば$n = 1$)に対して成り立つ
  - ある値nに対して成り立つことを仮定して$n + 1$でも成り立つことを示す。
  - 具体的に運用できること
  - 「・・・」は曖昧。使わない

グラフの記述$G = (V, A)$

- V：頂点の集合
- A：弧の集合
- 弧からその始点と終点への写像
  - $\partial^+ : A \to V,\ \partial^- : A \to V$
- 頂点から、そこを始点(終点)とする弧への写像
  - $\delta^+ : V \to 2^A,\ \delta^- : V \to 2^A$

特殊なグラフ

- 完全グラフ (complete graphs)
- 二部グラフ (bipartite graphs)
- 道 (paths)
- 閉路 (circles)
- 木 (trees)
- 極大木 (spanning trees)

二つの探索アルゴリズム
深さ優先探索

- 再帰的アルゴリズム
- 注目している頂点
- これまで探索を終えた頂点のリスト
- 再帰の動作に注意

$L = \{v_0\}$

$Q \longleftarrow \delta^+ v_0$ //push

while(Q $\neq \emptyset$) {

  $a = Q.\mathrm{pop}()$

  $w = \partial^- a$

  if ($w \notin L$) {

    $L \leftarrow L \cup \{w\}$

    U:wとLの要素を結ぶ弧

    $Q \longleftarrow \delta^+ w \setminus U$

  }

}

二つの探索アルゴリズム
幅優先探索

- 探索すべき頂点の待ち行列
- 再帰ではないことに注意

```
L = ∅
Q = [r]
while(Q ≠ ∅ ) {
    v = Q.poke//先頭
    forall( a ∈ δ⁺v ) {
        w = ∂⁻a
        if( w ∉ L && w ∉ Q) {
            Q ↤ w
        }
    }
    L ← L ∪ {v}
}
```

閉路探索

- Euler閉路
  - 一筆描き
  - 全ての弧を一度ずつ経由する閉路
- Hamilton閉路
  - 全ての頂点と一度ずつ経由する閉路
- 再帰的アルゴリズム
  - 閉路を列挙する

ネットワーク

- 弧に重み (実数) が定義されている
  - 距離
  - 費用
  - 容量
- 最適化問題
  - 最小木
  - 最短経路

最小木

- 弧の重みの和が最小になる極大木
- Kruskalアルゴリズム
  - 重みの小さい弧を順に採用
  - 閉路ができないように
- Jarník-Primアルゴリズム
  - 始点を定める
  - 木に取り込まれた頂点とそれ以外の頂点を結ぶ弧のうち、最小の重みの弧を追加：最小の増加

最小木
Kruskalアルゴリズム

$T = \emptyset$

H:Gは弧の重みに関するヒープ:最初に全弧を登録

while (TはGの極大木ではない){

　$a = H.\mathrm{poll}()$//ヒープから最小要素を取得

　$a_{\mathrm{new}} = \mathrm{null}$

　while (a_{new}==null) {

　　if ($T \cup \{a\}$ は閉路を持たない) $\{a_{\mathrm{new}} = a\}$

　　else　　$\{ a = H.\mathrm{poll}() \}$

　}

　$T = T \cup \{a_{\mathrm{new}}\}$

}

ヒープ使用は必須ではないが、ここでは最小重みの弧を探すためにヒープを利用

最小木
Jarník-Primアルゴリズム

任意の頂点$v \in V$を選び、$U = \{v\}$、$T = \emptyset$とする

while ($U \neq V$) {

　Uと$V \setminus U$を結ぶ弧のうち、最小の重みのものをaとする

　aの$V \setminus U$側の端点をwとする

　$U \leftarrow U \cup \{w\}$

　$T \leftarrow T \cup \{a\}$

}

Uは、Tを構成する頂点

最短経路

- 二つの頂点を結ぶ距離最小の有向道
- 始点を定め、全頂点への最短経路を求める

- Dijkstraアルゴリズム
  - $p: V \to R$：ポテンシャル関数
  - W：ポテンシャルが確定した頂点の集合
  - U：ポテンシャルを計算したが、未確定の頂点の集合
  - $q: V \to V$：最短距離に沿った直前の頂点

Dijkstra法：アルゴリズム

while ($U \neq \emptyset$) {

  $w = U.\mathrm{poll}()$ //$p(w)$が最小である$w \in U$

  forall ($a \in \delta^{+}w$) {

    $x = \partial^{-}a$ //wの隣接頂点

    if ($p(x) > p(w) + l(a)$) {//aを使ったほうが近距離

      $q(x) \leftarrow w$

      $p(x) \leftarrow p(w) + l(a)$

      if ($x \in U$) { $U.\mathrm{reduceValue}(x)$ //xの値を変更}

      else { $U.\mathrm{add}(x)$ //Uにxを追加}

  }}

  $W \leftarrow W \cup \{w\}$

}

ヒープ中の$p(x)$の値が
減ることがあることに注意

最大フロー

- 弧に容量とフローが定義されている
- アルゴリズム:頂点v_0から頂点v_dへの最大流量

流量を初期化する
補助ネットワークN_Aを作る
while (N_A中にv_0からv_dへの有向道がある){
　有向道に沿った増分dを求める
　N_Aを更新
}

補助ネットワーク更新アルゴリズム

update(N_A, P, d){
  foreach($a \in P$){
    $c_\varphi(a) \rightarrow c_\varphi(a) - d$
    if($c_\varphi(a) == 0$){aを削除}
    b：$a \in A_\varphi^{\pm}$ならば対応する弧$b \in A_\varphi^{\mp}$
    if(bが存在しない) {bを生成し、$c_\varphi(b) = d$}
    else {$c_\varphi(b) \rightarrow c_\varphi(b) + d$}
  }
}

元のネットワークへの反映アルゴリズム

N_A：補助ネットワーク
N：元のネットワーク

deploy(N_A){
  foreach ($a \in A$){//AはNの弧
    $b \in A_{\varphi}^{-}$：aに対応する弧
    $\varphi(a) = c(b)$
  }
}