アルゴリズム論（第8回）

2003.12.01
櫻井　彰人
algorithm@soft.ae.keio.ac.jp
http://www.sakurai.comp.ae.keio.ac.jp/

きょうの講義概要

- ハッシュ表（つづきのつもり. 実は最初）
- グラフ（1）
  - グラフの定義
  - クラスカル法
  - ダイクストラ法
- 原理、アルゴリズム、計算量、正当性、実際問題での利用

4.6 ハッシュ表の探索

ハッシュ表：
キーを引数とする算術式（ハッシュ関数）で格納位置を決定する表

例：
社員コード（x）4桁
ハッシュ関数：
int(x/100) + mod(x/100)
x=1424 ⇒ 14+24

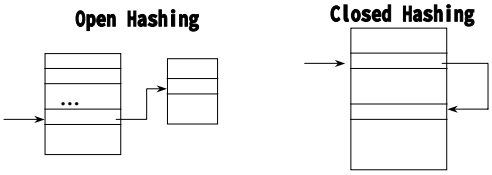
hash: 切り刻む (cf. hashed rice)　ばらばらにすることが肝要

ハッシュ関数

- ハッシュ関数 $h(x_k)$
  - 定義域：
    キーの値の最小値≦x_k≦キーの値の最大値
  - 値域：$1 \le h(x_k) \le$ ハッシュ表の大きさ
  - 計算時間が短い
  - 値が一様に分布（これが肝要）
  - 違うキーには異なる値（一般には不可能）⇐ かち合い（collision）がおこる

かち合いへの対応

- オープン・ハッシング
  - 予備の表（オーバーフローエリア）に
- クローズド・ハッシング
  - 同じハッシュ表内の空き領域に



アルゴリズム4.6 線形アドレシング（クローズド、検索）

- 入力：キーk、表の大きさMAX
- 出力：検索位置p
- 内容
  - （1）flag ← false; p← h(k); base ← p; i ← 1
  - （2）i < MAX である間以下を繰り返す
    - » (2.1) h(p) が探しているキーならば、flag ← true として（3）へ
    - » (2.2) p ← mod(p+1,M) ; i ← i+1
  - （3）flag = false なら、情報が見つからなかった。そうでなければ、p が格納位置

アルゴリズム4.7 クローズド・ハッシング（挿入）

- 入力：キーk、表の大きさMAX
- 出力：挿入位置i
- 内容
  - （1）flag ← false; p← h(k); base ← p; i ← 1
  - （2）i < MAX である間以下を繰り返す
    - » (2.1) h(p) が探しているキーならば、flag ← true として（3）へ
    - » (2.2) p ← mod(p+1,M) ; i ← i+1
  - （3）flag = false & i = MAX+1 ならば、ハッシュ表が一杯。そうでなければ、i に格納

探索時間の解析（1）

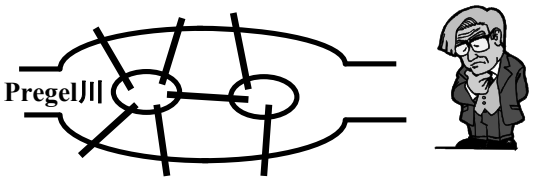
- 平均探索長
  - ハッシュ表の大きさ n、はいっているデータの個数 k
    - » かちあいが起こる確率 $p = k/n$
  - あるレコードが i回の探索で見つかる確率
    - » $p^{i-1}(1 - p)$
  - 平均探索長 $L = \Sigma_{i=1,k}\, i\, p^{i-1}(1 - p)$
    $\leq \Sigma_{i=1,\infty}\, i\, p^{i-1}(1 - p)$
    $= (1 - p)\, 1/(1 - p)^2$
    $= 1/(1 - p)$
    $= n/(n - k)$

探索時間の解析（2）

- $L = n/(n - k)$
  - n が k より十分大きいとき、線形探索の n/2 より小
- ハッシュ法
  - 探索速度がはやい
  - 表は固定長（表に空きができる）
  - ハッシュ関数に依存

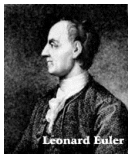
5.1 グラフ理論のおこり

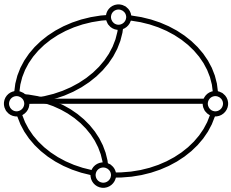
- Koenigsberg (Prussia) の7つの橋
  - 橋をちょうど1回ずつ通る経路？



「7つの橋」問題のグラフ表現

- 数学者オイラー（1707-1783）による解決
- 点（節, node, vertex）と線（辺, edge, arc）で問題を表現





岸

島

オイラーによる一般解

- 定義:
  - グラフは点（節）と互いに交わらない曲線（辺）とからなる図である.
  - 節は, もし奇数個の辺につながっているなら奇節, そうでなければ偶節と呼ぶ.
  - オイラー路とは, すべての弧をただ一度のみ通る途切れない路である.
- 定理:
  - グラフに3個以上の奇節があるなら, オイラー路は存在しない.
  - グラフにある奇節が2個以下ならば, 少なくとも一つのオイラー路がある.

5.3 諸定義

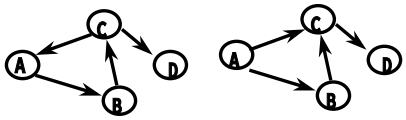
■ ラベル(label)
– 辺に付けた文字列や数値
» 文字列は節どうしの関係、数値はコストや距離

■経路 (path)
–特定の節v_iから節v_jへの辺の列
»$(v_i,a_1),(a_1,a_2),(a_2,a_3), \ldots, (a_n,v_j)$

諸定義(2)

■ 経路の長さ(length)
– 経路上で、始めと終わりの節に挟まれた節の個数(辺に距離のラベルがあれば、その総和)
» $(v_i,a_1),(a_1,a_2),(a_2,a_3), ..., (a_n,v_j)$の例ではn

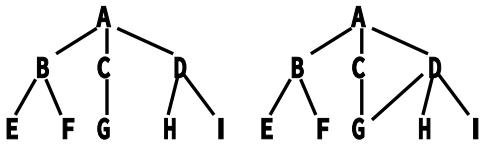
■ 閉路(cycleまたはloop)
– 同じ節に戻る経路



諸定義(3)

■ 単純路(simple path)
– 閉路を含まない経路

■ 木(tree)
– 閉路がないグラフ



グラフアルゴリズムの種類(例)

◆ あるグラフが存在
– 最小コストの全域木(全点を通る木)
» クラスカルのアルゴリズム
– 二つの節を最小コストでたどる
» ダイクストラ法
– すべての節間の最小コストの経路
» フロイトの方法
– 日程計画
» CPM (Critical Path Method)

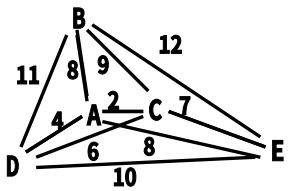
グラフアルゴリズムの種類(例2)

◆ あるグラフが存在(つづき)
– 平面性の判定
» 交差する辺がない平面グラフになるか?
– 節どうしの到達可能性
» 深さ優先探索でバックトラックしながら
– 閉路の発見
» 深さ優先探索でバックトラックしながら

◆ 二つ以上のグラフがあるとき
– 同形性の判定

5.5 クラスカルのアルゴリズム

◆ すべての地点を結ぶ木でコストが最小になるもの(コスト最小の全域木)を求める(道路網、電話網など)



考え方

- 節がn個で各辺にコストがついた無向グラフ G = {V,E}
- コストを最小に
  - 必ず木（全域木 S = {V,T}）に
    - » 木でない（閉路がある）なら、閉路の部分を（辺をとって）閉路でなくしても、つながりを保ったままコストが下げられる
- コストの小さい辺から順につなげる
  - つなげても効果のない（閉路になる）辺は無視
    - » 結果として、コスト最小に（あとで「証明」）

アルゴリズム5.1 クラスカル法

- 入力：節がn個で各辺にコストがついた無向グラフG = {V,E}
- 出力：コスト最小の全域木T
- 内容
  - （1）T ← Φ ; VS ← Φ
  - （2）Q ← コスト順に整列させた辺のリスト
  - （3）Vのすべての要素vに対して、{v} を要素とする集合VSを作る

クラスカル法（2）

（4）‖VS‖＞1の間、以下を繰り返す

(4.1) Qの先頭から辺(u,w)を取り出し、これをQから取り除く

(4.2) uとwが、VSの集合の要素として異なる集合W1とW2に属するときだけ以下を実行

(4.2.1) VS中のW1とW2を取り除き、そのあとW1∪W2を追加する

(4.2.2) T←T∪{(u,w)}

クラスカル法の実行例

Qの要素（コストの小さい順、＜辺、コスト＞）：
<(A,C),2>, <(A,D),4>, <(C,D),6>, <(C,E),7>, <(A,B),8>, <(A,E),8>, <(B,C),9>, <(D,E),10>, <(B,D),11>, <(B,E),12>

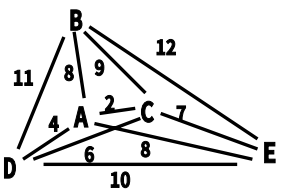
アルゴリズム5.1の実行過程：

辺	Tへの処理	VSの要素
-	-	{A}, {B}, {C}, {D}, {E}
(A,C)	追加	{A,C}, {B}, {D}, {E}
(A,D)	追加	{A,C,D}, {B}, {E}
(C,D)	×	上のまま
(C,E)	追加	{A,C,D,E}, {B}
(A,B)	追加	{A,B,C,D,E}

例題の実行結果

- できあがるコスト最小の全域木

T = { (A,C), (A,D), (C,E), (A,B) }



クラスカル法の正当性

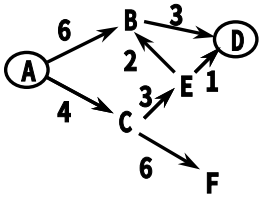
- クラスカル法で得られる全域木$T = \{t_1, t_2, ..., t_n\}$なら、コスト最小の極大木$T_0 = T$
  - （略証）以下の手順でT_0をTに変換する手続きを考え、T_0とTが異なる木だと矛盾することを示す
    - » （1）k ← 1
    - » （2）T_0に含まれない $(x,y) = t_k$ があれば、$T_0 \cup \{t_k\}$ を作り、この中にできた閉路上の辺でTに含まれない辺(v,w)を選び、$T_0 \leftarrow (T_0 - \{(v,w)\}) \cup \{(x,y)\}$
    - » （3）k ← k + 1
    - » （4）$k \leq n$ なら（2）へ。そうでなければ終了

クラスカル法の計算量

- n個の節があって、辺の個数mのグラフ
  - はじめにVSを作る
    - » n回の操作
  - 辺の整列
    - » m log m
  - 辺を追加するかどうかの比較
    - » m以下
  - 辺を追加する集合操作（高速なものがあると仮定）
    - » m以下
  - 結局、m log m

5.6 ダイクストラ法

- 各辺に距離のついた有向グラフ上で特定の2点間の最短距離（最終的にはすべての節への最短距離）



考え方：
出発の節から近い節を順次たどる。結果的には、指定された出発の節から全部の節への最短経路を求められる。

アルゴリズム5.2 ダイクストラ法

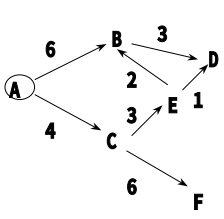
- 入力：各辺に距離のついた有向グラフ、出発の節
- 出力：出発の節からすべての節への最短距離
- 内容
  - （1）U ← 空集合； V ← すべての節の集合； s ← 出発の節； dist(s) ← 0, dist(u) ← ∞ (u ≠ s)
  - （2）V が空集合でない間、以下を繰り返す
    - » (2.1) p ← Vの要素で dist が最小の節
    - » (2.2) p を V から除き、U に加える
    - » (2.3) p から直接結ばれている節すべてに対して以下を繰り返す
      - (2.3.1) x ← そのうちで一つ適当に選ばれた節
      - (2.3.2) x がVに属していれば、dist(x) ← min(dist(x),dist(p)+この辺の距離)

ダイクストラ法（2）

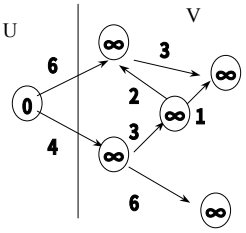
- ダイクストラ法の計算量
  - （2）と（2.3）のところで2重のループ
    - » 多めに見積もって $\Theta(n^2)$

実行例（1）

- 与えられたグラフと出発の節

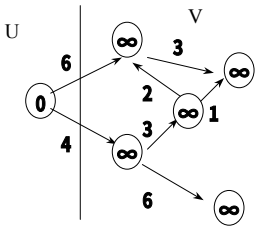


Step 0

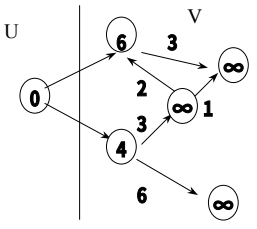


Step 1 (2.2の直後)

実行例（2）

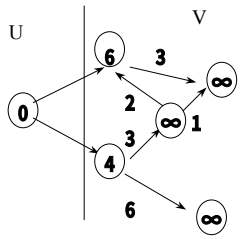


Step 1 (2.2終了時)

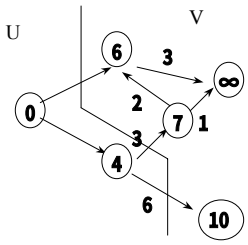


Step 2 (2.3終了時)

実行例(3)

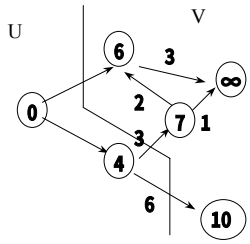


Step 2 (2.3終了時)

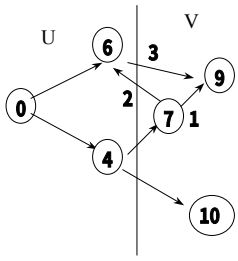


Step 3 (2.3終了時)

実行例(4)

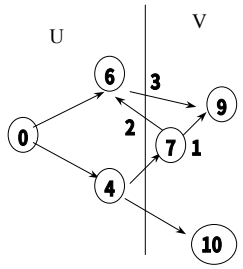


Step 3 (2.3終了時)

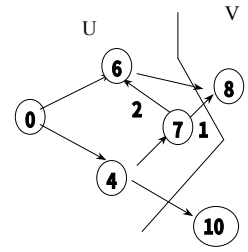


Step 4 (2.3終了時)

実行例(5)

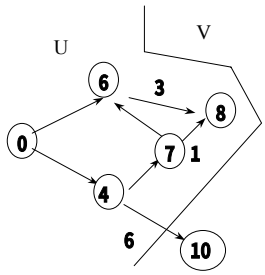


Step 4 (2.3終了時)

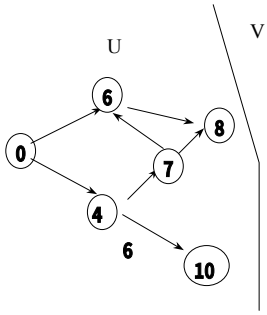


Step 5 (2.3終了時)

実行例(6)



Step 6 (2.3終了時)



ダイクストラ法の正当性

- **(a) sからの距離と経路がすでに確定した各節uに対して、dist(u)がsからuへの最短距離であり、uまでの最短経路は確定した節だけから成っている**
- **(b) まだ経路が確定していない各節uに対しては、dist(u)がsからuへの任意の経路で最短の距離(そのような経路がないときは∞)**