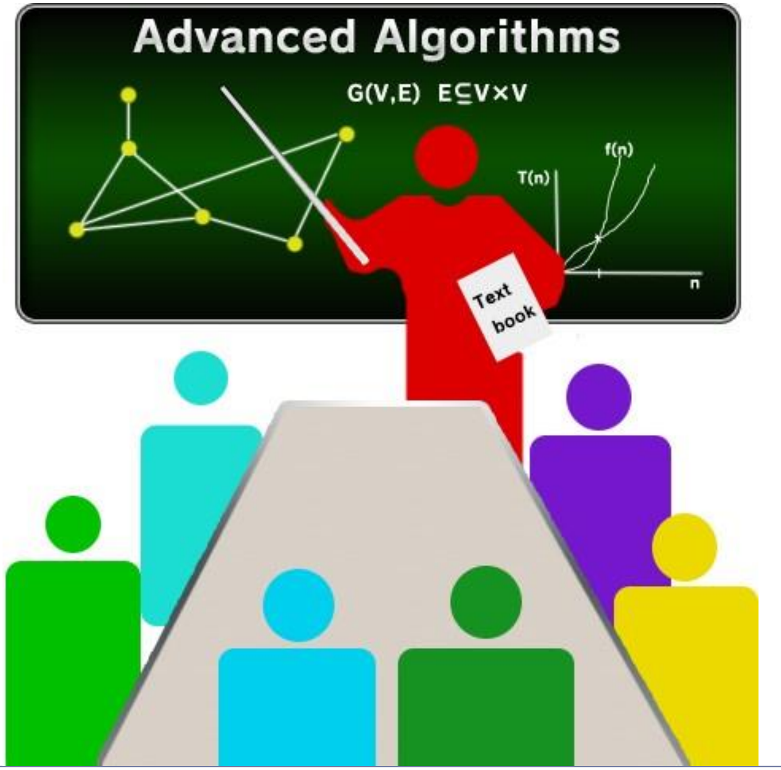


アルゴリズム特論 [AA201X] Advanced Algorithms

Lecture03. Graphs and their representation

Exercise 03 のために

- グラフの基礎
- 表現方法（隣接行列、隣接リスト）
- 全域木
- Depth First Search,　Breath First Search
- 関節点

グラフ・集合の基礎

- G(V, E)
- グラフGは、**頂点の集合V**と**辺の集合E**で構成される

- 「集合」の復習
- $x \in A$ （元 xは集合Aに含まれている， x inA ）
- $A \cap B$（積集合：A,Bのどちらにも含まれる要素の集合）
- $A \cup B$（和集合：A,Bのどちらか少なくとも一方に含まれる要素の集合）
- $A \cup \{x\}$ （集合Aに Aの元ではない元 x を加えた集合 ）
- $A - B$（差集合：Aに含まれる要素からBに含まれる要素を取り去った集合）
- **集合の直積** $V_1 \times V_2$（V_1とV_2に含まれる要素のペアの全パターンの集合。
  例： $A=\{a,b,c\}, B=\{1,2\}$のとき$A \times B=\{ (a,1), (a,2), (b,1), (b,2), (c,1), (c,2) \}$
  ※集合Vに対して、**V^2と書くと$V \times V$、つまり集合V自身同士の直積となる**
- $|A|$ （集合Aに含まれる元の数、上の例だと$|A|=3, |B|=2$である ）

グラフ・集合の基礎

- **有向グラフ（Directed Graph　）**

⇒ 辺の向きの区別があるグラフ

頂点 a から b への辺　と　頂点 b から a への辺は別物として区別する

＝ Eの要素は <u>順序対</u> である

- **無向グラフ（Non-directed Graph）**

⇒ 辺の向きの区別がないグラフ

頂点 a から b への辺　と　頂点 b から a への辺は同一視する

＝ Eの要素は <u>非順序対</u> である

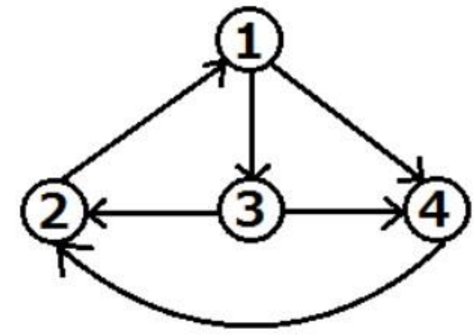
グラフの表現方法

- **隣接行列 ： O(|V|²) の空間計算量**

頂点 i から j へ向かう

- 道がある：$A_{i,j} = 1$
- 道がない：$A_{i,j} = 0$

$$\begin{array}{cc} & \text{j へ} \\ \text{i から} & \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$



有向グラフ $G(V, E)$

- **隣接リスト ： O(|V|+|E|) の空間計算量**
- 連結リストで実装。頂点数分のリストを作成
- 各頂点から繋がっている頂点だけを接続する

（C++ならvector / list で実装可能）

- 隣接行列が疎行列なら有理
- 密行列の場合は不利

頂点

1 ⇒ 3 ⇒ 4

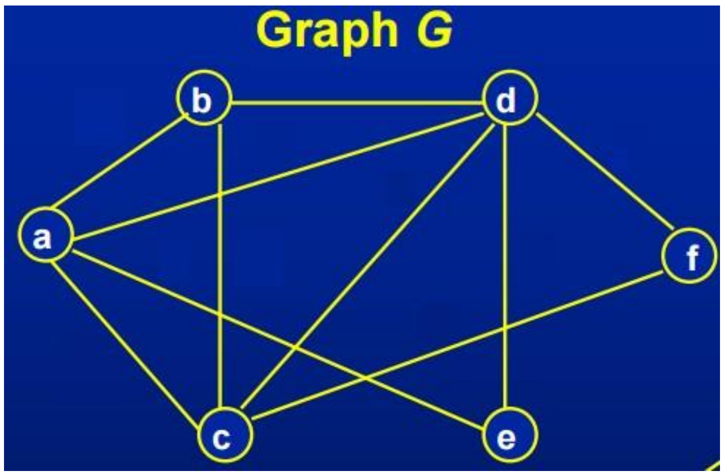
2 ⇒ 1

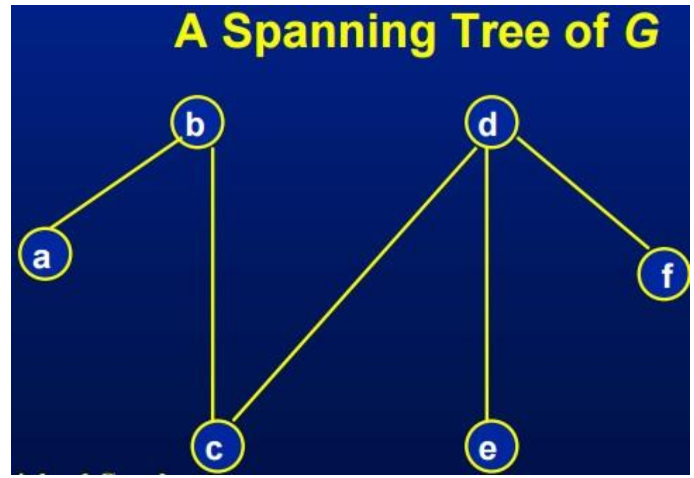
3 ⇒ 2 ⇒ 4

4 ⇒ 2

全域木（Spanning Tree）

- グラフG(V,E) 上の 辺 E の部分集合 T が 木になっている（閉路を持たない）の T





- 閉路 （cycle）

⇒ 始点からグラフ上の経路を辿って行くと、再び始点に辿り着けるような道（頂点の列）。

全域木の検出方法

- 深さ優先探索（Depth First Search）

・道が繋がっているならば、進めるだけ進んでいく
・道が無くなったら、他に進める道があるところまで
　戻って探索を再開する
⇒ Backtracking の考え方　(Ex11)

再帰やスタックで実装

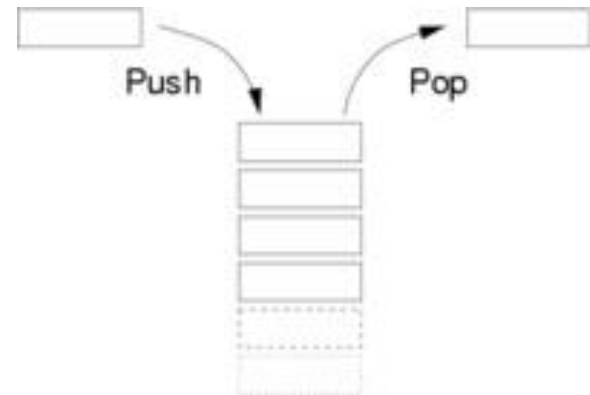
- 幅優先探索（Breath First Search）

・ある頂点を始点としたとき、始点から辿って同じレベル（深さ、距離と考えた方が分かりやすいか？）の頂点から順に辿る
・始点から距離１の頂点を全部探索
　⇒距離２の頂点を全部探索　⇒　距離３の頂点を…

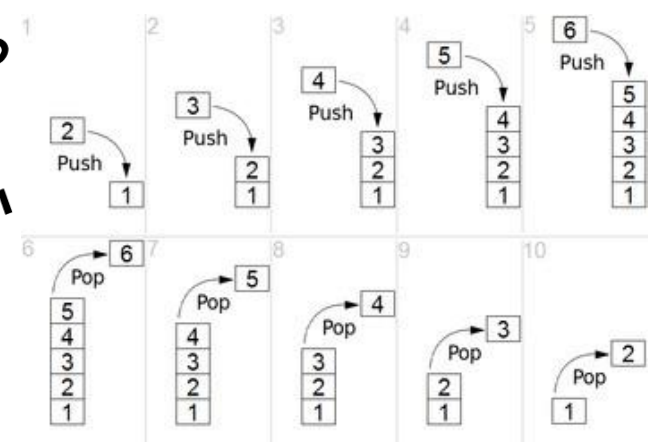
キューで実装

スタック（FILO: First In Last Out）

- 底にフタのある入れ物に上から
データを入れたり取り出したりする
・最後に入れたものが最初に取り出せる



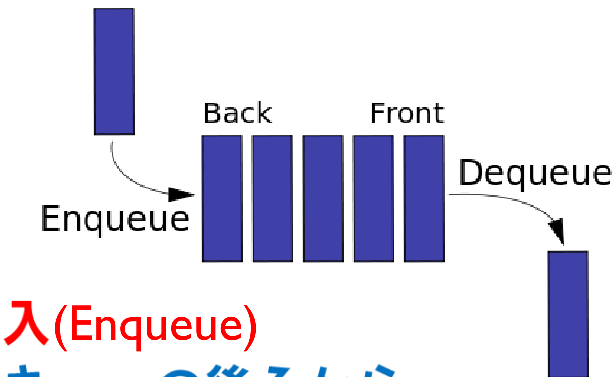
- DFS実装のための使い方
・DFSで探索（到達）した頂点をスタックにPushする
・もう探索ができなくなったらスタックからPopする
・Popした頂点から未探索頂点へ辿れるか調べる
・探索再開できる頂点が見つかるまでPopする
・スタックが空 ＝ もう探索はこれ以上できない



キュー（FIFO: First In First Out）

- **待ち行列：順番待ちの列のような構造**

・先に入ったものは先に出る
・後に入ったものは先に入ったものが全て出ていくまで待つ



- **BFS実装のための使い方**

・BFSで探索（到達）した頂点をキューの先頭に挿入(Enqueue)
・BFSで同じレベル内で探索が不可能になったら、キューの後ろから頂点を取り出す（Dequeue）
・取り出した頂点から探索を再開する（＝次のレベル内を探索）
・キューが空　＝　もう探索はこれ以上できない
・右図のBackとFrontは、スケルトンコードの head と tail に相当する
これをキューの状態をこれを使って配列でどう表現するかがポイント

関節点（Articulation Points）

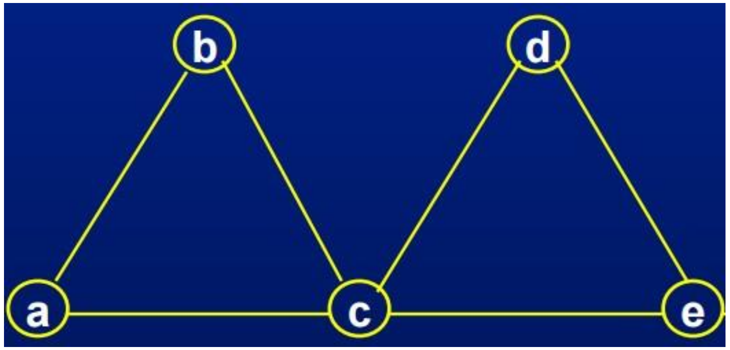
- **k-（頂点）連結グラフ　（k-connected graph）**

・グラフGから任意の k-1 個の頂点を取り去った**部分グラフが連結グラフの状態を維持している**

・1-connected graph

k-1 = 1-1 = 0　、つまり１個でも頂点を取り除くと連結グラフは崩壊してしまう。

⇒ その原因が **関節点** (Articulation Point)



例：右図での関節点の集合は A={ c }

・2-connected graph

⇒ 任意に１個の頂点を取り除いても連結グラフのまま

⇒ 任意に２個の頂点を取り除いたとき、連結性は崩壊

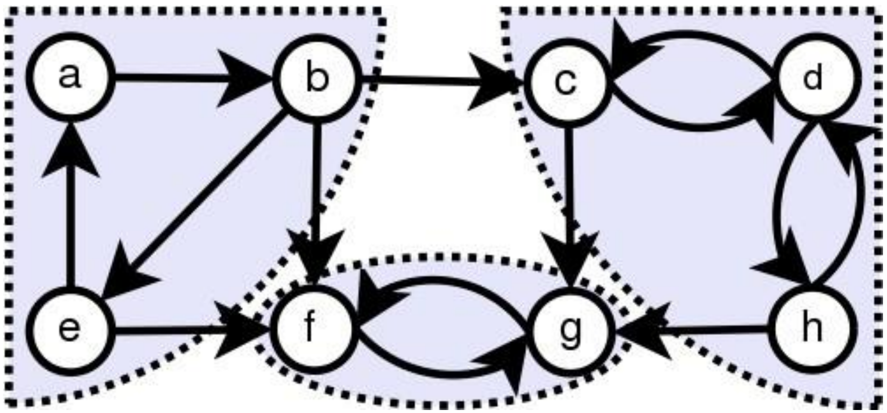
強連結（Strongly connected）

- 有向グラフ G(V,E) 上の2つの頂点 u,v が 強連結

⇔ u から v へ辿る方法がある && v から u へ辿る方法がある

右図においては、V = {a,b,c,d,e,f,g,h}

{a,b,e}, {f,g}, {c,d,h} が強連結成分



⇒強連結成分分解 (Strongly Connected Components Decomposition)

・実装方法としては…

DFSを２回使えば解ける（ボーナス問題）