データ構造とアルゴリズム
（第14回）

モバイルコンピューティング研究室
柴田史久

1

本日の講義内容

- グラフ（教科書未掲載，テスト範囲内）
  - グラフとは
  - グラフの表現
    - 隣接行列，隣接リスト
  - 探索
    - 深さ優先探索，幅優先探索，最小全域木
  - 最短経路問題
    - ダイクストラのアルゴリズム
- バックトラック法
- 動的計画法

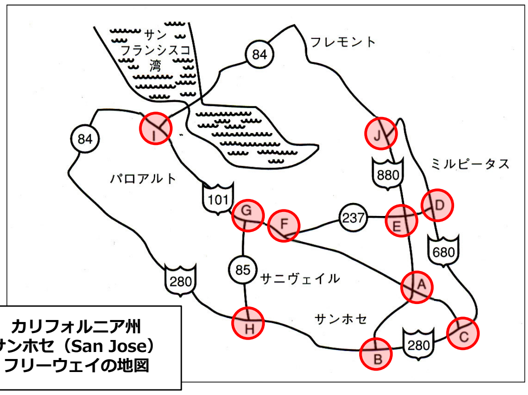
2

教科書 未掲載
参考文献4：第13章グラフ，第14章重み付きグラフ
参考文献3：第3章グラフのアルゴリズム

グラフ

3

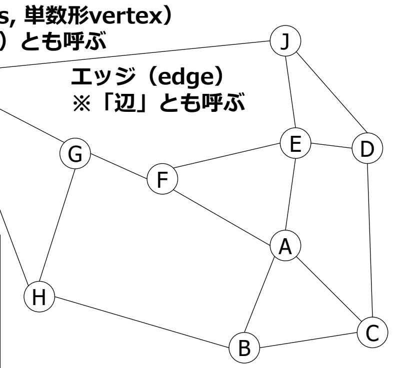
ロードマップ



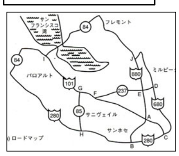
カリフォルニア州
サンホセ（San Jose）
フリーウェイの地図

4

グラフで表現すると



グラフは現実世界の
具体的な状況を表現



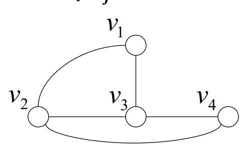
5

グラフとは(1)

- グラフ：Graph
- グラフ G は頂点（vertex）の有限集合 V と頂点の対を結ぶエッジ（edge）の有限集合 U で定義

$$G = (V, U)$$

- エッジは頂点対 v_i, v_j を用いて (v_i, v_j) と表現



- 個々の物事とそれらの接続関係を抽象的に表現

6

グラフとは(2)

- **隣接**
  - **1本のエッジの両端にある頂点は，隣接している（adjacent）と表現．ある頂点に隣接している頂点のことを近傍（neighbors）と呼ぶ**
- **路（パス，path）**
  - **ある頂点から他の頂点までに連なる一連のエッジのこと．ロードマップの例では B – A – E – J などが該当**
- **連結グラフ（connected graph）**
  - **各頂点から他のすべての頂点へパスが１つ以上あるグラフのことを連結している（connected）と表現**

7

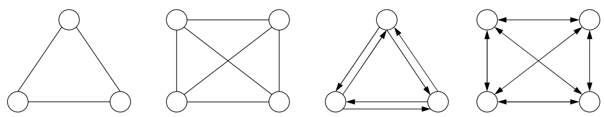
グラフとは(3)

- **無向グラフ（non-directed/undirected graph）**
  - **エッジに方向性（direction）がないもの**
- **有向グラフ（directed graph）**
  - **エッジに方向性があるもの．図示する場合にはエッジの先端に矢印を描く**
    - **ex. 一方通行の道路，仕事の順序**
- **重み付きグラフ（weighted graph）**
  - **エッジに重みのついたグラフ．図示する場合にはエッジの横に重みの数値を書く．**
    - **ex. 都市間の距離，電車の運賃**

8

グラフとは(4)

- **頂点の数：** n
- **エッジの数：** m
- **無向グラフでは最大** $n(n - 1) / 2$
- **有向グラフでは最大** $n(n - 1)$
- **エッジの数最大のグラフ：完全グラフ（complete graph）**

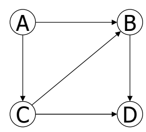


無向グラフ　　**有向グラフ**

9

グラフの表現(1):頂点

- **最も単純化するなら番号で表現することも可能**
- **実際には頂点に情報を格納**



対象とする有向グラフの例

```
class Vertex {
  // ラベル．例えば '1'，'A' など
  public char     label ;
  // 訪問済みを意味するフラグ
  public boolean wasVisited ;
  // 必要なら他のフィールドもつくる

  public Vertex(char l) {
    label      = l ;
    wasVisited = false ;
  }
}
```

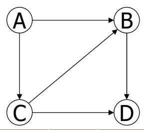
10

グラフの表現(2)

- **隣接行列（adjacency matrix）**
- n **次の正方行列：** A
- **エッジ** (v_i, v_j) **が存在するときに** i **行** j **列の要素** a_{ij} **を** $\boxed{1}$ **とし，存在しない場合に** 0 **とする**

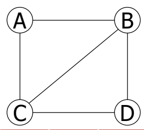
補足：
1ではなくエッジの本数を設定することで多重エッジも表現可

有向グラフ



	A	B	C	D
A	0	1	1	0
B	0	0	0	1
C	0	1	0	1
D	0	0	0	0

無向グラフ

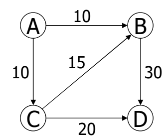


	A	B	C	D
A	0	1	1	0
B	1	0	1	1
C	1	1	0	1
D	0	1	1	0

11

グラフの表現(3)

- **重み付きグラフの場合**
  - **通常は隣接行列の他に重み行列が必要**
  - **エッジが存在しないことを無限大などの特別な値で表現すれば隣接行列のみで表現可能**

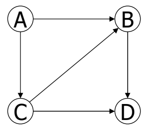


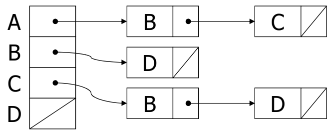
	A	B	C	D
A	∞	10	10	∞
B	∞	∞	∞	30
C	∞	15	∞	20
D	∞	∞	∞	∞

12

グラフの表現(4)

- 隣接リスト（adjacency list）
- 各頂点を始点とする辺のリストを頂点毎に作成
- リスト１本１本を隣接リストと呼ぶ
- リスト中の頂点の並び方とグラフのパスは無関係

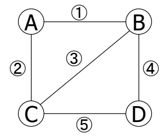




13

グラフの表現(5)

- 接続行列（incidence matrix）
- $n \times m$ の行列
- 頂点 v_i，エッジ $e_k = (v_i, v_j)$ とする
- 頂点 v_i がエッジ e_k に接続している場合に，i 行 k 列の要素 a_{ik} を 1 とし，存在しない場合に 0 とする



無向グラフ

	①	②	③	④	⑤
A	1	1	0	0	0
B	1	0	1	1	0
C	0	1	1	0	1
D	0	0	0	1	1

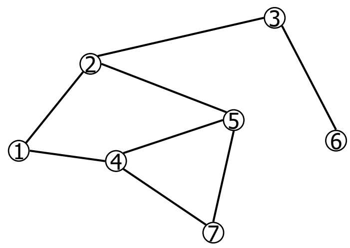
14

グラフの探索(1)：深さ優先探索

- Depth first search, 縦型探索
  - 一つの道を選んでいけるところまで行き，進めなくなったら引き返して別の道を選ぶ探索法
  - スタックを用いて実現
- 手順
  - 最初の頂点を訪問
  - そこから出る辺を一つ選びその先の頂点を訪問
  - この頂点からも同様に辺を選び先の頂点を訪問
  - 以下，同様に行き止まりまで進む
    - 行き止まり＝訪問済み頂点，辺のない頂点
  - 行き止まりになったら引き返して調べてない辺の先の頂点を訪問
  - これを繰り返しすべての頂点から出る辺を探索すれば終了

15

深さ優先探索の実行例



$m = n - 1$

n : 頂点の数
m : エッジの数

最小全域木：頂点同士が最小のエッジで接続されたグラフ

- １つのグラフに可能な最小全域木は複数存在
- エッジの長さは考慮外
- 深さ優先探索のときにたどったエッジを記録すれば最小全域木

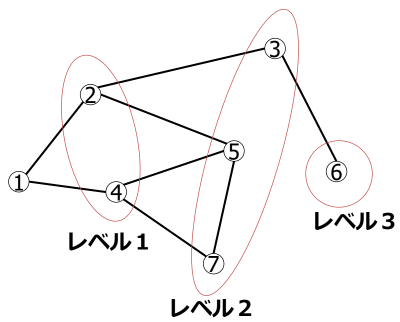
16

グラフの探索(2)：幅優先探索

- Breadth first search, 横型探索
  - 開始頂点から距離的になるべく近い頂点を探す探索法
  - キューを用いて実現
- 手順
  - 最初の頂点を訪問
  - この頂点から到達可能な頂点（レベル１頂点）をすべて訪問
  - レベル１頂点のいずれかから到達可能な頂点（レベル２頂点）を訪問
  - 以上を，繰り返す
  - 一度訪問した頂点は二度と訪問しない

17

幅優先探索の実行例



18

最短経路問題

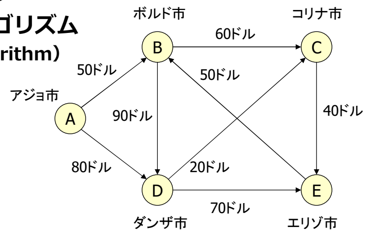
- 重み付きグラフを対象として２つの頂点間の最短パスを見つける問題
  - 最短の定義：経路を構成するエッジの重みの和が最小
  - 非常に応用範囲の広い問題
    - 料金最小
    - 時間最短

19

具体例:鉄道路線

- ジャングルの中の単線（エッジは有向）
- 都市間の料金は固定（遠距離割引などはない）
- 条件：コストは非負

- 最短経路問題の解法
  - ダイクストラのアルゴリズム
    - 貪欲法（greedy algorithm）



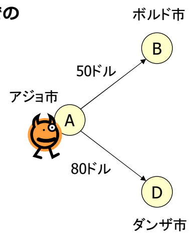
20

ダイクストラのアルゴリズム(1)

アジョ(A)から	ボルド(B)	コリナ(C)	ダンザ(D)	エリゾ(E)
Step1	50(アジョ経由)	∞	80(アジョ経由)	∞

ルール：出発点（アジョ）からの合計料金が最も安い都市へエージェントを送りこむ

※注：エージェントの仕事は
その都市から隣接都市までの
料金情報を集めること



21

ダイクストラのアルゴリズム(2)

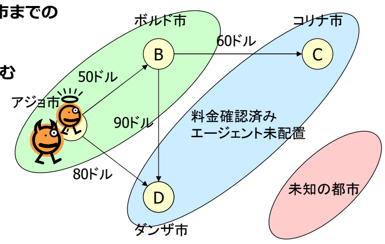
アジョ(A)から	ボルド(B)	コリナ(C)	ダンザ(D)	エリゾ(E)
Step1	50(アジョ経由)	∞	80(アジョ経由)	∞
Step2	50(アジョ経由)	110(ボルド経由)	80(アジョ経由)	∞

ダンザにはボルド経由でも到達可能だが最安ルートに興味があるから高い料金のルートは無視

エージェントを送り込んだ都市までの
最短経路は確定

ボルドにエージェントを送り込む

エージェント
配置済み



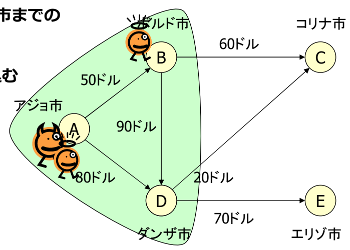
22

ダイクストラのアルゴリズム(3)

アジョ(A)から	ボルド(B)	コリナ(C)	ダンザ(D)	エリゾ(E)
Step1	50(アジョ経由)	∞	80(アジョ経由)	∞
Step2	50(アジョ経由)	110(ボルド経由)	80(アジョ経由)	∞
Step3	50(アジョ経由)	100(ダンザ経由)	80(アジョ経由)	150(ダンザ経由)

エージェントを送り込んだ都市までの
最短経路は確定

ダンザにエージェントを送り込む



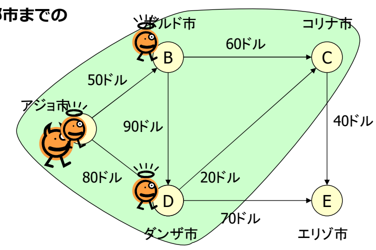
23

ダイクストラのアルゴリズム(4)

アジョ(A)から	ボルド(B)	コリナ(C)	ダンザ(D)	エリゾ(E)
Step1	50(アジョ経由)	∞	80(アジョ経由)	∞
Step2	50(アジョ経由)	110(ボルド経由)	80(アジョ経由)	∞
Step3	50(アジョ経由)	100(ダンザ経由)	80(アジョ経由)	150(ダンザ経由)
Step4	50(アジョ経由)	100(ダンザ経由)	80(アジョ経由)	140(コリナ経由)

エージェントを送り込んだ都市までの
最短経路は確定

コリナにダンザ経由で
エージェントを送り込む



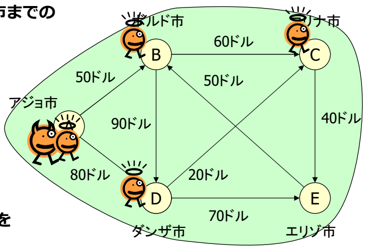
24

ダイクストラのアルゴリズム(5)

アジョ(A)から	ボルド(B)	コリナ(C)	ダンザ(D)	エリゾ(E)
Step1	50(アジョ経由)	∞	80(アジョ経由)	∞
Step2	50(アジョ経由)	110(ボルド経由)	80(アジョ経由)	∞
Step3	50(アジョ経由)	100(ダンザ経由)	80(アジョ経由)	150(ダンザ経由)
Step4	50(アジョ経由)	100(ダンザ経由)	80(アジョ経由)	140(コリナ経由)
Step5	50(アジョ経由)	100(ダンザ経由)	80(アジョ経由)	140(コリナ経由)

エージェントを送り込んだ都市までの最短経路は確定

エリゾにコリナ経由でエージェントを送り込む



すべての都市にエージェントを送り込んだら終了

25

ダイクストラのアルゴリズム(6)

- ポイント
  - エージェントを新しい都市に送るたびに，そのエージェントから得られた情報で料金表を改訂
  - 出発地点からある都市までの最安の料金だけ記載
  - 新しいエージェントを送るのは出発点からの最安経路上の都市

26

教科書 第19章（pp.437〜458）

バックトラック法

27

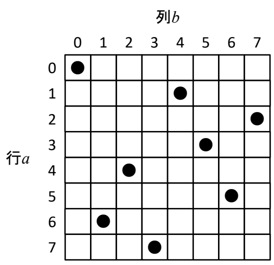
バックトラック法

- Backtracking，後戻り法
- すべてのパターンを系統的に探索して解答を得る
  - 行けるところまで進み，ダメだったら戻るを繰り返す
  - 試行錯誤の途中でこれ以上先に進んでもムダだと判明したら，後戻りして別の選択肢に進む
- 再帰で実現
- ポイントはできるだけ早めに選択肢を絞り込むこと

28

具体例:8クイーン問題

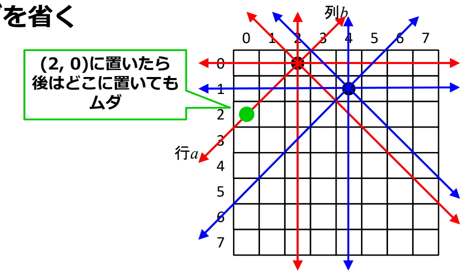
- 8×8のチェス盤上に，8つのクイーンをたがいに利き筋に当たらないように配置する
  - チェスのクイーンは将棋の飛車・角を合わせた動きをする



29

8クイーン問題の考え方

- 何も考えないクイーンの置き方
  - 64×63×. . . ×57 = 178,462,987,637,760通り
- クイーンは各行，各列に1つずつしか置けない
  - 8! = 40,320通り
- さらにムダを省く



30

バックトラック法の実現

- 手順
  - 最初のクイーンを行0の適当な場所に置く
  - 行0のクイーンの利き筋を避けて行1にクイーンを置く
  - 行0，1のクイーンの利き筋を避けて行2にクイーンを置く
  - 以下，同様に行7までクイーンを置く
  - 途中で利き筋を避けてクイーンを置けなくなったら，1行前のクイーンを別の場所に移動して再トライ
  - 1行前のクイーンについて可能なすべての置き場所を使ったなら，2行前のクイーンを別の場所に移動して再トライ

31

8クイーンの解法(疑似コード)

詳細は教科書 p.443 List 19.1

```
// メソッド tryQueen は行a以降のクイーンをすべて置ければtrueを返す
boolean tryQueen(int a) {
  for (場所(a,0), (a,1), …… , (a,7) について繰り返す) {
    if (この場所は他のクイーンの利き筋ではない) {
      この場所にクイーンを置く
      if (a == 7) {             // すべてのクイーンが置けた
        return true ;
      } else {
        if (tryQueen(a+1))      // 行a+1以降のすべてに置けた
          return true ;
        else
          失敗したのでクイーンを盤から取り除く
      }
    }
  }
  return false ; // 行aにはクイーンを置ける場所がなかった
}
```

32

教科書 第20章（pp.459～476）

動的計画法

33

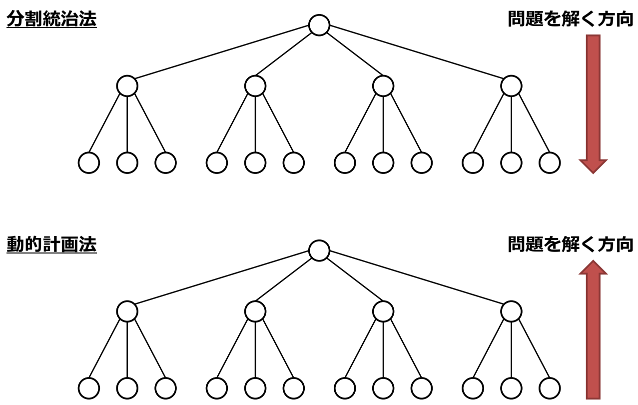
動的計画法(dynamic programming)

- 以下の条件を満たすアルゴリズムの総称
  - 分割統治法
    - 問題を部分問題に分割し，各個撃破する手法
  - メモ化
    - 部分問題の計算結果を再利用する
- 利用するための条件
  - 問題を部分問題に分割でき，部分問題の解の組み合わせで元の問題を解けること
  - 分割された部分問題の数が多すぎないこと

参考文献：Wikipedia 「動的計画法」

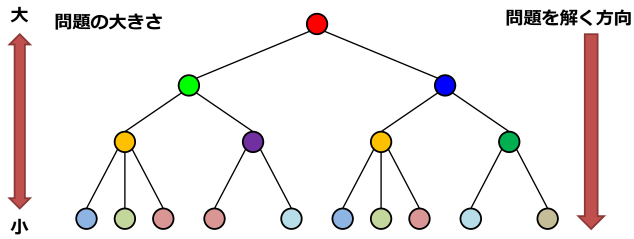
34

分割統治法と動的計画法



35

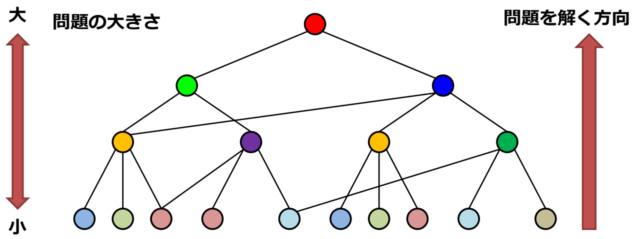
分割統治法



分割統治法では与えられた大きな問題を細かく分割して，最終的にごくささいな問題に帰結させる（トップダウン）

36

動的計画法



動的計画法では，まずはじめにささいな問題を解決して，
その結果を積み上げながら最終的な結果を得る（ボトムアップ）

37

まとめ

- グラフ（教科書未掲載）
  - グラフとは
  - グラフの表現
    - 隣接行列，隣接リスト
  - 探索
    - 深さ優先探索，幅優先探索，最小全域木
  - 最短経路問題
    - ダイクストラのアルゴリズム
- バックトラック法
- 動的計画法

38

参考文献

1. 定本 Javaプログラマのための
   アルゴリズムとデータ構造（近藤嘉雪）
2. 新・明解 Javaで学ぶ
   アルゴリズムとデータ構造（柴田望洋）
3. 岩波講座ソフトウェア科学 3
   アルゴリズムとデータ構造（石畑清）
4. Javaで学ぶアルゴリズムとデータ構造
   Robert Lafore（著）・岩谷 宏（翻訳）
5. Java アルゴリズム+データ構造完全制覇
   オングス (著)・杉山 貴章・後藤 大地 (監修)

39