プログラミング応用 第7回

河瀬 康志

2018年7月30日

アウトライン

演習問題 (1/2)

問 1

容量 100 と 1000 の 2 つの仮定の下で下記のナップサック問題を解き，最適解での価値と，使われるアイテムの名前のリストを出力せよ．ただし各行は「重さ 価値 名前」が並んでいるものとする．

http://yambi.jp/lecture/advanced_programming2018/vegetables.txt

問 2

Dijkstra 法を最短経路で通る頂点も出力するように修正せよ．また，以下のグラフの 1 から 5 までの最短経路を出力せよ．

```
adj = {1:[(2,7),(3,2),(4,6)],
       2:[(4,8),(5,5)],
       3:[(4,4)],
       4:[(5,7)],
       5:[]
       }
```

演習問題 (2/2)

問 3 (おまけ)

数字のリスト a が与えられた時，$i < j$ ならば $a[i] < a[j]$ を満たす部分列で (増加部分列) 最長のものの長さを求めよ．

`http://yambi.jp/lecture/advanced_programming2018/prob6-3.txt`

例: $a = [4, 2, 8, 3, 5, 1, 7]$

問 4 (おまけ)

左上から右下までの道（上下左右のみ移動可能）で，数字の和が最小のものを求め，その値を答えよ．

`http://yambi.jp/lecture/advanced_programming2018/prob6-4.txt`

例：
$$\begin{pmatrix} 131 & 673 & 234 & 103 & 18 \\ 201 & 96 & 342 & 965 & 150 \\ 630 & 803 & 746 & 422 & 111 \\ 537 & 699 & 497 & 121 & 956 \\ 805 & 732 & 524 & 37 & 331 \end{pmatrix}$$

授業スケジュール

	日程	内容
第 1 回	6/11	ガイダンス・復習
第 2 回	6/18	文字列操作（文字列整形，パターンマッチ，正規表現） 平面幾何（線分の交差判定，点と直線の距離，凸包）
第 3 回	6/25	乱数（一様分布，正規分布への変換，乱数生成） 統計（データ処理，フィッティング）
第 4 回	7/2	計算量（オーダー表記） スタックとキュー（幅優先探索，深さ優先探索）
第 5 回	7/9	ソートアルゴリズム バックトラック（N クイーン問題，数独）
第 6 回	7/23	動的計画法（ナップサック問題） 最短経路探索（Warshall-Floyd, Bellman-Ford, Dijkstra）
第 7 回	7/30	巡回セールスマン問題
期末試験	8/6	南 4 号館 3 階 第 1 演習室で実施

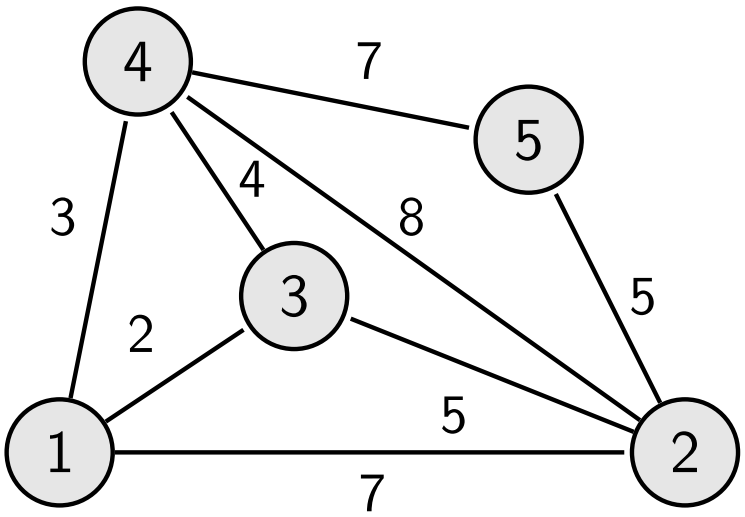
期末試験について

- 日時： 8/6(月) 13:20 から
- 場所： 南 4 号館 3 階 第 1 演習室
- 試験時間は 80 分
- 持ち込み可
  - 授業資料はパソコンにダウンロードしておいてください.
  - 宿題の解答を手に入れておくことをおすすめします.
    (おまけ問題は試験には出しません)
  - 本などの持ち込みも OK
  - 人の持ち込みは不可
- インターネットの使用は不可
  (携帯，スマホ禁止．パソコンの wifi はオフ)

アウトライン

巡回セールスマン問題とは

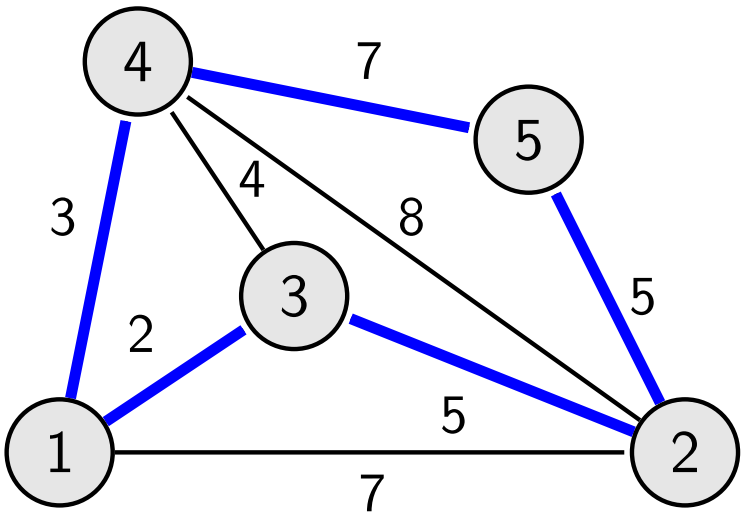
全頂点をちょうど 1 回ずつ通って 1 周する最短ルートを求める問題



- 難しい問題 (NP 困難)
- 扱い方
  - 時間を掛けてがんばる
  - ある程度の解で我慢する
    - 保証が欲しい
      ⇒ 近似解法
    - 何でもいい
      ⇒ ヒューリスティクス
      (発見的解法)

巡回セールスマン問題とは

全頂点をちょうど 1 回ずつ通って 1 周する最短ルートを求める問題



- 難しい問題 (NP 困難)
- 扱い方
  - 時間を掛けてがんばる
  - ある程度の解で我慢する
    - 保証が欲しい
      ⇒ 近似解法
    - 何でもいい
      ⇒ ヒューリスティクス
      (発見的解法)

ユークリッド巡回セールスマン

- 点が d 次元空間中に配置されていて，2 点間の距離はユークリッド距離である問題
- 今回は特に 2 次元の場合を扱う

```
import random
n=128
ps = [(random.uniform(50,750),random.uniform(50,550)) for i in range(n)]

def dist(s,t): return ((s[0]-t[0])**2+(s[1]-t[1])**2)**(0.5)
def total_dist(ps,order):
    n = len(ps)
    return sum([dist(ps[order[i]], ps[order[(i+1)%n]]) for i in range(n)])
```

ブルートフォース

```
import itertools
# 全探索
def brute_force(ps):
    min_dist = float('inf')
    res = []
    for a in itertools.permutations(range(len(ps))):
        d = total_dist(ps,a)
        if min_dist > d: min_dist, res = d, a
    return res
```

```
# 短く書き直した版
def bf(ps):
    return min([a for a in itertools.permutations(range(len(ps)))],key=(lambda x:total_dist(ps,x)))
```

- 全部の順番 ($n!$ 通り) を調べ，最小長さのものを求める
- 計算時間は $O(n \cdot n!)$
- 10 点でもかなり時間がかかる

動的計画法

- 適当に v を固定し，v から出発して v に戻ると思ってよい
- 次の条件を満たす最短経路長を $\mathrm{table}[u][S]$ とする $(u \notin S \subseteq V)$
  - v から出発し，その後 S の頂点をちょうど 1 回ずつ訪れる
  - 最後に u へ到着する
- 以下の漸化式が成立

$$\mathrm{table}[u][S] = \min_{w \in S} \{\mathrm{table}[w][S \setminus \{w\}] + \mathrm{dist}(w, u)\}$$

- 計算時間は $O(n^2 \cdot 2^n)$
- 16 点くらいまでは計算できる

動的計画法 — Python による実装

```
def dp(ps):
    length = {} # length[(u,S)]: u を始点とし S の点すべてを回る最小経路長
    route = {} # route[(u,S)]: 最小経路長を達成するためのルート
    v = ps[0]
    n = len(ps)
    for i in range(1,n+1):
        for a in itertools.combinations(range(n),i):
            S = frozenset(a)
            for j in S:
                u = ps[j]
                if i==1:
                    length[(j,S)] = dist(v,u)
                    route[(j,S)] = [j]
                else:
                    Sj = S-set([j]) # S から j を除いたもの
                    k=min(Sj,key=lambda k: length[(k,Sj)]+dist(ps[k],u))
                    length[(j,S)] = length[(k,Sj)]+dist(ps[k],u)
                    route[(j,S)] = route[(k,Sj)]+[j]
    return route[(0,frozenset(range(n)))]
```

参考： メモ化再帰

メモ化再帰を用いても同様に書ける

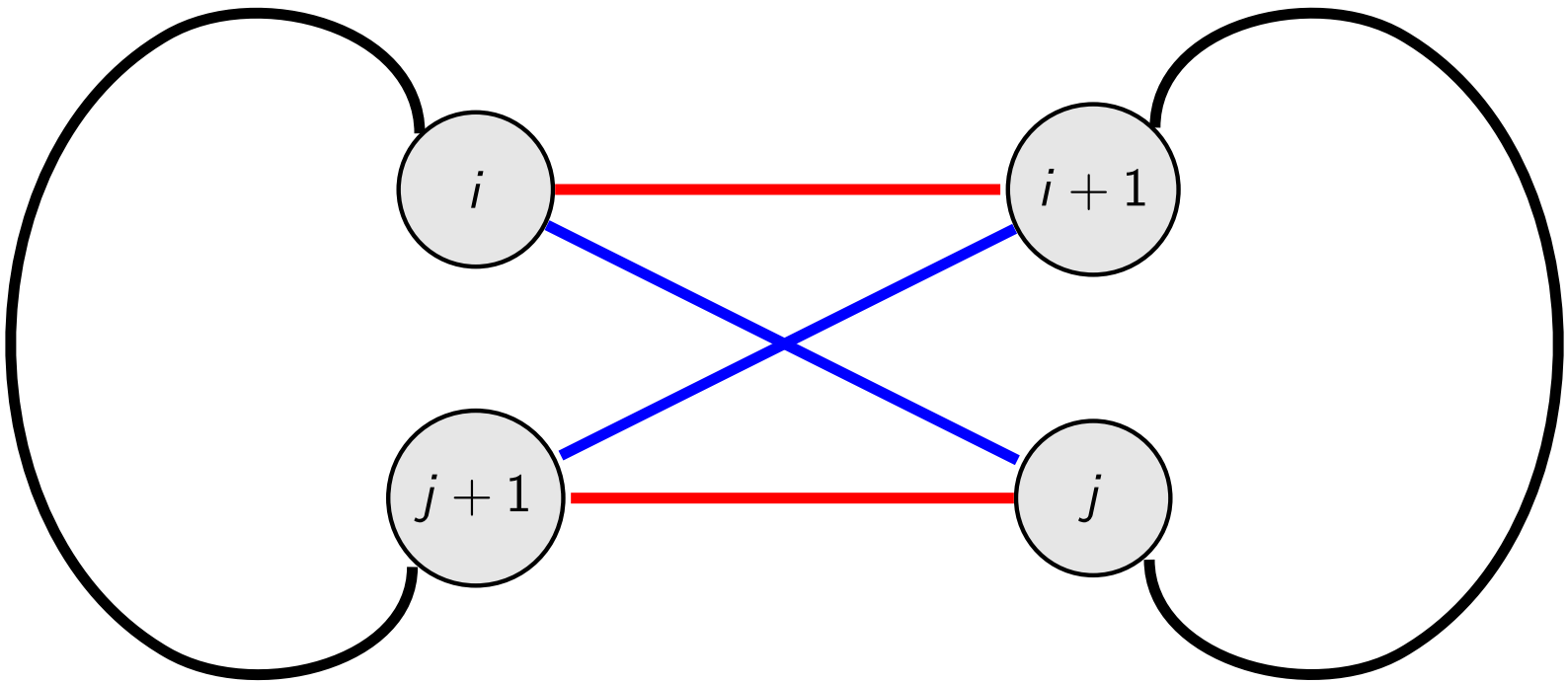
```
def dp_memo(ps):
    return _dp_memo(ps,0,0,frozenset(range(1,len(ps))),{})[1]+[0]
def _dp_memo(ps,i,j,S,memo):
    if (i,j,S) in memo: return memo[(i,j,S)]
    if len(S)==0: return (dist(ps[i],ps[j]),[])
    # Si = frozenset([x for x in S if x!=i])
    Si = S-set([i])
    k=min(S,key=lambda k:_dp_memo(ps,k,j,S-set([k]),memo)[0]+dist(ps[i],ps[k]))
    (d,l)=_dp_memo(ps,k,j,S-set([k]),memo)
    memo[(i,j,S)]=(d+dist(ps[i],ps[k]),[k]+l)
    return memo[(i,j,S)]
```

ヒューリスティクス

- 発見的解法，Heuristic
- 理論的な保証はないが実験的には良い解が得られる方法
- 今回はその中でも近傍探索法を用いる
  - 適当な解を作る
  - 現在の解の「近傍」内によりよい解があれば解を置き換える
  - よりよい近傍解が存在しなくなれば終了（局所最適解）

2-opt

- 近傍として枝 2 本を付け替えた解を考える
- $\mathrm{dist}(i, i+1) + \mathrm{dist}(j, j+1)$ と $\mathrm{dist}(i, j) + \mathrm{dist}(i+1, j+1)$ を比較する
- 後者がより小さければ，$i+1$ 番目から j 番目の点を逆順にする



2-opt — Python による実装

```
def two_opt(ps):
    n = len(ps)
    res = list(range(n))
    update = True
    while update:
        update = False
        print(total_dist(ps,res))
        for (i,j) in itertools.combinations(range(n),2):
            if ((dist(ps[res[i]],ps[res[i+1]])+dist(ps[res[j]],ps[res[(j+1)%n]]))>
               (dist(ps[res[i]],ps[res[j]])+dist(ps[res[i+1]],ps[res[(j+1)%n]]))):
                res[i+1:j+1] = res[j:i:-1]
                update = True
    return res
```

ヒューリスティクスでさらに良い解を得るには

- 最初の解を何にするか
  - 今回： 最初に与えられたまま
  - ランダムな順番
  - まだ訪れていない最も近い点へ次々と移動して得られる解（貪欲解）
- 近傍の定義
  - 今回： 2 枝の交換のみ (2-opt)
  - 3 枝，4 枝の交換 (3-opt,4-opt,k-opt)
  - 連続する k 枝の入れ替え
- 近傍の移動戦略
  - 今回：改善する近傍が見つかればすぐに移動 (即時移動戦略)
  - 近傍の中で最も良い解に移動 (最良移動戦略)
  - たとえ悪くなっても移動する (アニーリング法，タブー探索法)
- その他
  - 局所探索を何度も繰り返す（多スタート，反復，可変近傍）
  - 遺伝アルゴリズムなど別の方法

Advanced Topic: 精度保証付き近似アルゴリズム

精度保証付き近似アルゴリズム

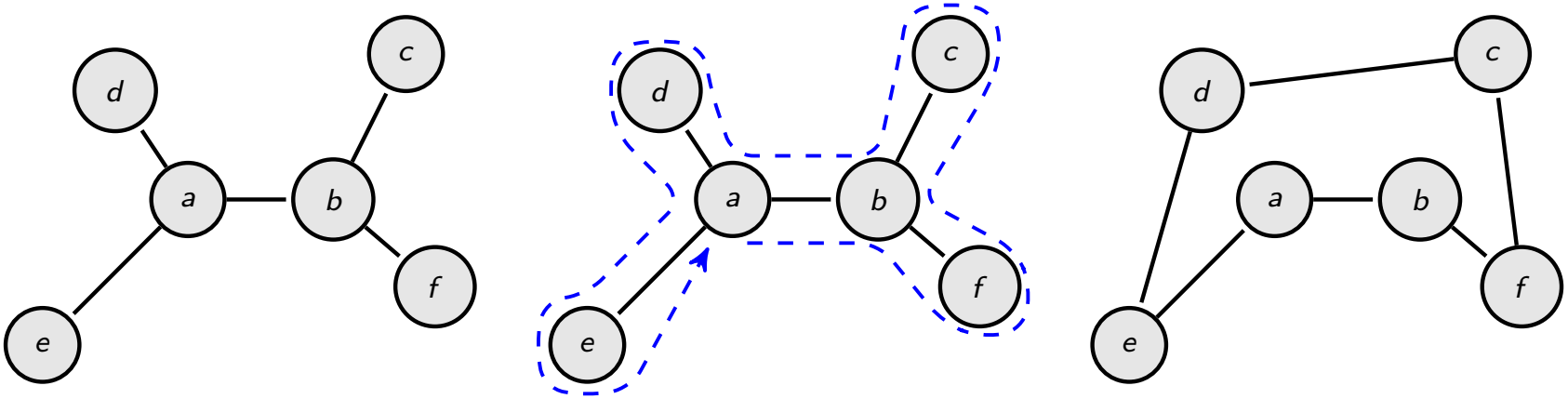
- アルゴリズムの出力する解の目的関数値と最適値の比がある範囲内に入ることが保証されているもの
- (最小化問題に対する)α-近似アルゴリズム

$$\text{アルゴリズムの出力する解の目的関数値} \leq \alpha \cdot \text{最適値}$$

- α は 1 以上であり，小さいほどうれしい．$O(\log n)$ のように問題サイズ n に依存する場合もある
- 多項式時間アルゴリズムであることを求められることが多い

Advanced Topic: TSP に対する 2 近似アルゴリズム

- 最小全域木を求める
  - 閉路を含まず，全ての点がつながっているような枝部分集合
  - 効率的に計算できる
- 最小全域木を深さ優先探索し，行きがけ順で点を訪れる



Advanced Topic: 最小全域木の求め方

Prim 法

- 動的計画法の一種
- アルゴリズム：
  - 適当に始点 s を決め $X = \{s\}$ とする
  - X と $V \setminus X$ をつなぐ最小コストの枝を e とし，解に追加
  - e でつながれている点を X に追加
  - $X = V$ となるまで繰り返し
- 計算量は $O(|E| \log |V|)$
- Dijkstra 法とほぼ同様（ヒープに入れる距離が違うだけ）

実装例

```
def mst(ps):
    n = len(ps)
    #prim
    mst = {v:[] for v in range(n)} # MST の隣接リスト
    X = set()
    heap = [] # (dist,u,v)
    for v in range(1,n):
        heapq.heappush(heap,(dist(ps[0],ps[v]),0,v))
    while len(heap)>0:
        (d,u,v)=heapq.heappop(heap)
        if v not in X:
            X.add(v)
            mst[u].append(v)
            mst[v].append(u)
            for w in range(n):
                if w not in X: heapq.heappush(heap,(dist(ps[v],ps[w]),v,w))
    #dfs
    res = []
    stack = [0]
    visited = [False]*n
    while len(stack)>0:
        v = stack.pop()
        if visited[v]: continue
        visited[v]=True
        res.append(v)
        for u in mst[v]: stack.append(u)
    return res
```

アウトライン

演習問題提出方法

解答プログラムをまとめたテキストファイルを作成して，OCW-i で提出

- ファイル名は practice7.txt
- 期末試験の開始時間が締め切り
- ファイルの最初に学籍番号と名前を書く
- どの演習問題のプログラムかわかるように記述
- 出力結果もつける
- 途中までしかできなくても，どこまでできてどこができなかったかを書けば部分点を付けます

演習問題

問 1

以下のように作成したランダムな 1000 点に対して，ユークリッド巡回セールスマン問題を考え，長さ 26 以下の解があることを確認し，どのように確認したかを記せ．
(実は長さ 23.1 以下の解が存在)

```
import random
n=1000
random.seed(0)
ps = [(random.random(),random.random()) for i in range(n)]
```

問 2 (おまけ)

内半径 50mm のパイプに半径が 50,49,48,47,...,40mm のボールを入れるとき，パイプの長さは何ミリメートル必要か．整数値で答えよ．