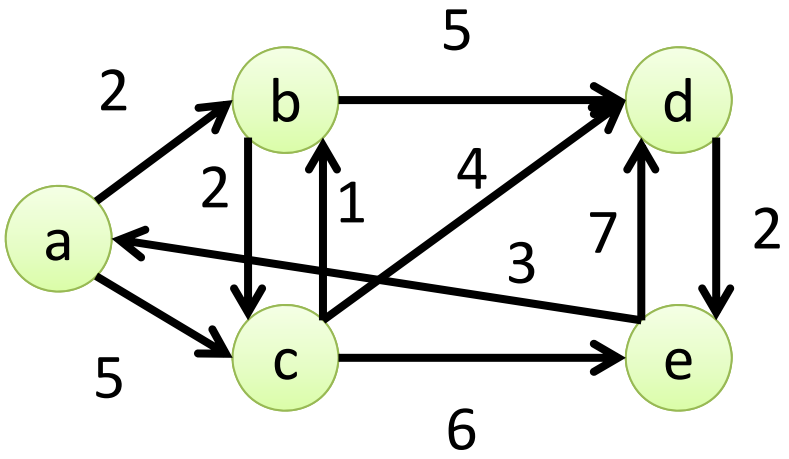
アルゴリズムとデータ構造
第13回: 最短経路問題

担当: 上原 隆平(uehara)

2014/05/27

最短経路問題

- 入力: 辺重み付きグラフ，始点s，終点t
- 出力: 始点sから終点tへの路のうち，辺重み和最小のもの



Q. aからeの最短経路は?

A. a→b→d→e

- 負の重みの有無で問題の難しさが変わる
- 本講義では重さは全て正であると仮定

最短経路問題のバリエーション

- 単一頂点対最短経路問題: s から t へ
  始点 s から 終点 t への最短経路を求める
- 単一始点最短経路問題: s から
  始点s から他の全頂点への最短経路を求める
- 単一目的地最短経路問題: t へ
  終点t への他の全頂点からの最短経路を求める
- 全頂点対最短経路問題: すべてのペア間
  全頂点対 (s, t) について最短経路を求める

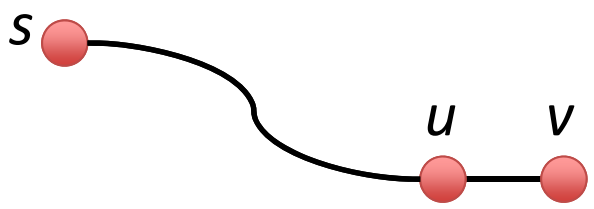
最短経路の部分構造の最適性

- 定理: グラフ G で，$P=(v_1, v_2, \ldots, v_k)$ を v_1 から v_k への最短経路とする．このとき任意の i と j $(1 \leqq i \leqq j \leqq k)$ に対し，$P_{i,j}=(v_i, v_{i+1}, \ldots, v_j)$ は v_i から v_j への最短経路である．

- 証明: $P_{i,j}$より短い路 $P'_{i,j}=(v_i, w_1, \ldots, w_h, v_j)$ があるとすると，$P'=(v_1, \ldots, v_i, w_1, \ldots, w_h, v_j, \ldots, v_k)$ は P より短い v_1-v_k 路となり，P の最短性に矛盾．

最短経路の部分構造の最適性

- 系: グラフ G における始点 s から頂点 v への最短経路 $P_{s,v}$ が s から 頂点 u までの最短路 $P_{s,u}$ と辺 (u, v) で構成されるとき，つまり，$P_{s,v}=P_{s,u} + (u,v)$ のとき，$w(P_{s,v}) = w(P_{s,u}) + w(u,v)$.
  - $w(u,v)$: 辺 (u,v) の重み
  - $w(P)$: 路 P に含まれる辺の重み和

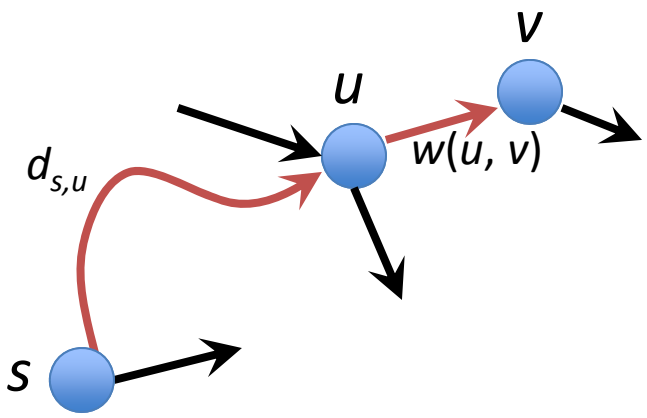


最短経路の求め方: 緩和法

緩和法:
始点 s から各頂点 v までの距離を $d[v]$ として推定し，
この推定値を徐々に真の値に近づけていく方法

補題: グラフ G において，始点 s から頂点 v までの距離（最短路の長さ）を $d_{s,v}$ と表すとき，全ての辺 (u, v) に対して，

$$d_{s,v} \leqq d_{s,u} + w(u, v)$$



遠回りしても近くならない
cf. 三角不等式

最短経路の求め方: 緩和法

緩和法:
始点 s から各頂点 v までの距離を $d[v]$ として推定し，
この推定値を徐々に真の値に近づけていく方法

緩和操作 (relaxation):
辺 (u, v) に対する緩和操作では，これまでに発見されている
v への最短路を u を通ることにより改善できるかどうかを判定

```
relax(u, v)
  if d[v] > d[u] + w(u, v) then
    d[v] = d[u] + w(u, v);
    vの先行点 = u;
```

注意!

- d[ ] は最初すべて無限大に初期化しておく
- 正しい順序で頂点を調べないと真の値にならない.

緩和法の性質 (1/4)

補題1: グラフ G の任意の辺 (u, v) に対して緩和 relax(u, v) を実行した後では，$d[v] \leqq d[u] + w(u, v)$ が成立

証明: relax(u, v) を実行する前に $d[v] \leqq d[u] + w(u, v)$ が成り立っていれば relax(u, v) では何もしないので成立.

逆に $d[v] > d[u] + w(u, v)$ であれば，relax(u, v) において $d[v] := d[u] + w(u, v)$ を実行するので成立.

緩和法の性質 (2/4)

補題2: グラフ G と始点 s が与えられて，s 以外の全ての頂点 v について $d[v] = \infty$ に初期化したとき，緩和操作を繰り返しても不等式 $d[v] \geqq d_{s,v}$ は常に成立

証明: 初期状態では $d[v] = \infty$ より $d[v] \geqq d_{s,v}$ となり成立.
緩和操作 relax(u, v) により，初めて $d[v] < d_{s,v}$ となったとする.
relax(u, v) で $d[v]$ の値が更新されているので，その直後では，

$$d[u] + w(u, v) = d[v] < d_{s,v} \leqq d_{s,u} + w(u, v)$$

が成立. relax(u, v) で $d[u]$ の値は更新していないので，
relax(u, v) の前に $d[u] < d_{s,u}$ が成立していたことになり，矛盾.

緩和法の性質 (3/4)

補題3: グラフ G と始点 s が与えられているとする．始点 s から頂点 v への最短路を $(s, \ldots, u, v)$ とする．relax(u, v) を含む緩和操作を繰り返し実行するとき，relax(u, v) を実行する以前に $d[u] = d_{s,u}$ であれば，この呼出の後で $d[v] = d_{s,v}$ が成立．

証明: 補題2より，relax(u, v) を実行する前に $d[u] = d_{s,u}$ ならば relax(u, v) 実行後も $d[u] = d_{s,u}$．また，補題1より，relax(u, v) の実行後は次式が成立:

$$d[v] \leqq d[u] + w(u, v) = d_{s,u} + w(u, v) = d_{s,v}.$$

補題2より，$d[v] \geqq d_{s,v}$ は常に成り立つから $d[v] = d_{s,v}$.

緩和法の性質 (4/4)

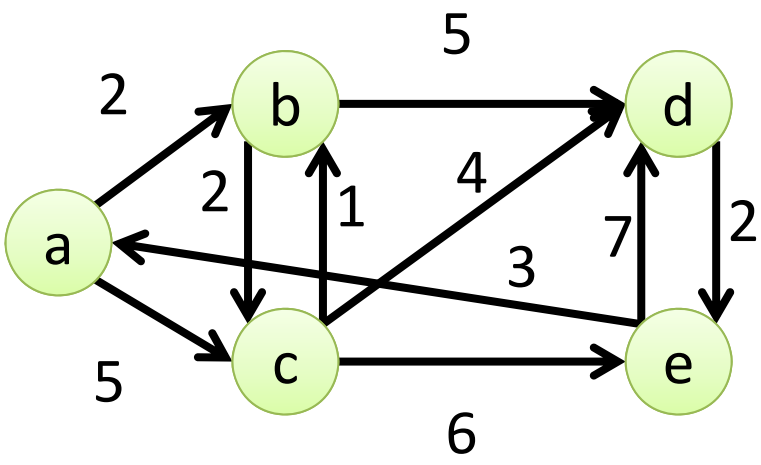
補題4：グラフ G と始点 s が与えられているとする．初期化した後では，先行点部分グラフは s を根とする木を形成する．この性質は G の辺を任意の順に選んで緩和操作を施しても保たれる．

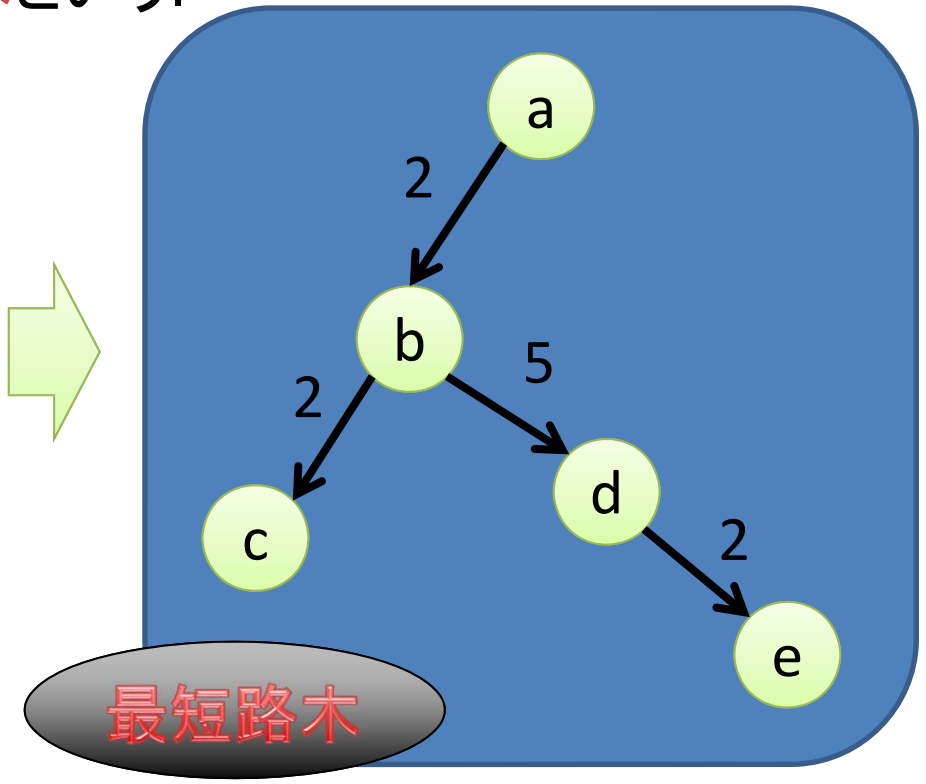
先行点部分グラフとは，$d[]$ が有限となった頂点，つまり，先行点が決まった頂点（と始点 s）からなるグラフ．辺は頂点からその先行点へと向き付けされる．

証明… の前に

最短路木

グラフ G と始点 s が与えられたとき各頂点 v について s から v への最短路における v の先行点を $p(v)$ とする.
辺 $(p(v), v)$ だけからなるグラフは常に木となるが,
これを s を根とする最短路木という.





緩和法の実現: 初期化・緩和操作

- 初期化操作
  - 各頂点 v について
    - 始点からの距離: $d[v] = \infty$
    - 先行点: $p(v) = \text{nil}$
  - 始点 s については $d[s] = 0$
- 緩和操作

初期化の直後では先行点部分グラフは s のみからなるグラフ

```
relax(u, v)
  if d[v] > d[u] + w(u, v) then
    d[v] = d[u] + w(u, v);
    p(v) = u;
```

  - 緩和操作のたびに辺が 1 本ずつ増加するが，閉路はできない

緩和法の性質 (4/4)

補題4: グラフ G と始点 s が与えられているとする．初期化した後では，先行点部分グラフは s を根とする木を形成する．この性質は G の辺を任意の順に選んで緩和操作を施しても保たれる．

証明: 緩和操作の繰り返し回数に関する帰納法による

- Base case: 初期化直後: 自明
- Inductive case: 緩和操作を n 回繰り返して得られたグラフが閉路を持たない s を根とする木であるとき，もう1回緩和操作を実行し，
  - 閉路が作られた
  - s を根とする木でなくなった

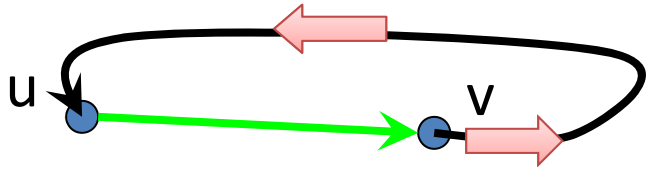
  として矛盾を導く（背理法）．

緩和法の性質 (4/4): 閉路がないこと

```
relax(u,v)
  if d[v]>d[u]+w(u,v) then
    d[v]=d[u]+w(u,v);
    p(v)=u;
```

補題4: グラフ G [illegible] では，先行点部[illegible] 性質は G の辺を任意の順に選んで緩和[illegible] も保たれる．

Inductive caseの証明: 緩和操作を n 回繰り返して得られたグラフが閉路を持たないとき、n+1 回目の操作で辺 (u, v) が作られて閉路ができたとする．

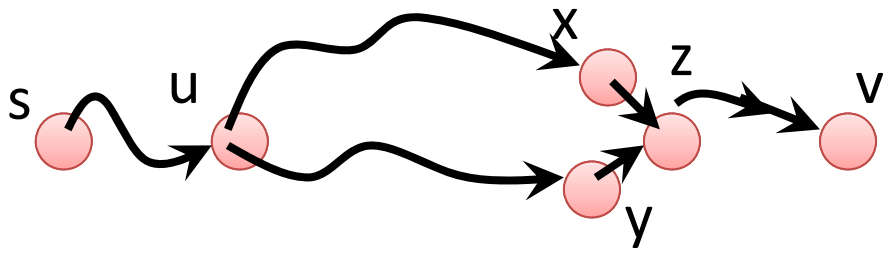


その直前では先行点部分グラフは v から u へのパスを含んでいたはずだから $d[v] < d[u]$ が成り立っているはずである．すべての辺は正の重みをもつので，$d[v] < d[u] + w(u, v)$ が成立．これは辺 (u, v) に関して緩和操作が行われたことに矛盾する。

緩和法の性質 (4/4): s を根とする木

補題4: グラフ G と始点 s が与えられているとする．初期化した後では，先行点部分グラフは s を根とする木を形成する．この性質は G の辺を任意の順に選んで緩和操作を施しても保たれる．

Inductive case: s からのパスが見つからない限り，$d[v]$ は有限にならないので，先行点も設定されない．よって，根は s．木でないとすると，根からの経路が1通りでないような頂点 v が存在する．



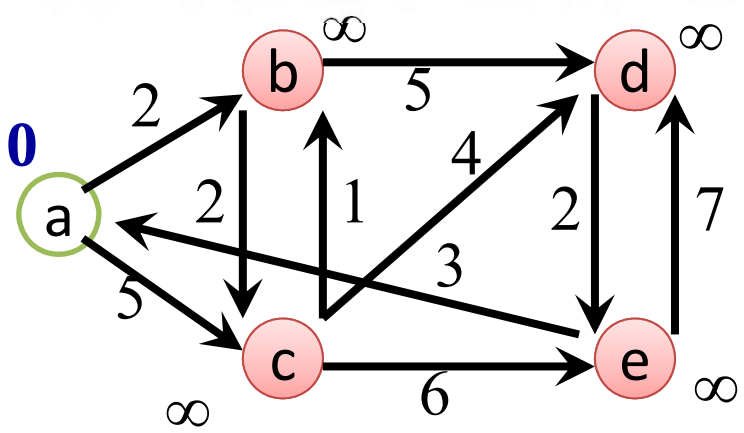
2通り以上の経路があるなら逆に辿ると必ず分岐点がある。分岐点は先行点を2個以上持たなければならないが、これはどの頂点も先行点はただ一つしかないという性質に矛盾。

最短経路の求め方: ダイクストラ法

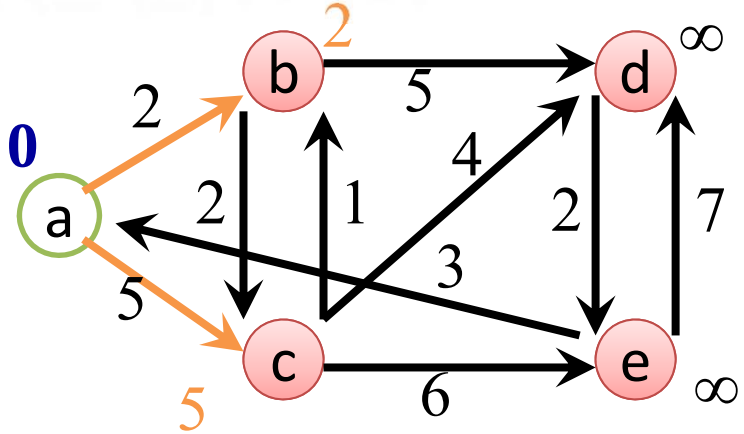
始点 s からの距離 $d_{s,v}$ が求まった頂点の集合を S とし，V–S の中から，最短路推定値 $d[u]$ が最小である頂点 u を選び，u を S に追加し，u から出る全ての辺に対して緩和操作を行う．これを繰り返す．

```
for v∈V do d[v]=∞; p(v) = nil; end
d[s]= 0; S=∅;
while(V–S ≠ ∅) do
  u = argmin_{v∈V–S} d[v];
  S = S∪{u}
  for (u,v)∈E do relax(u,v); end
end
```

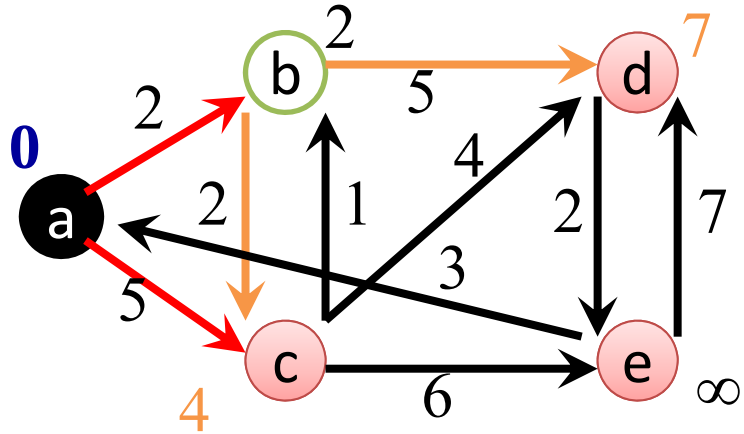
ダイクストラ法の実行例(1/2)



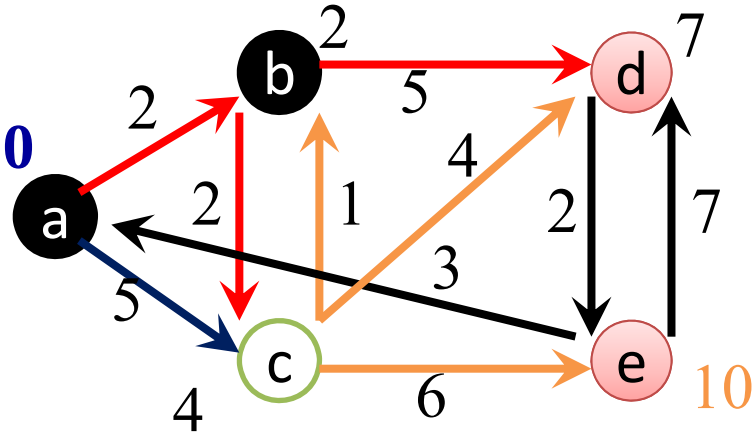
1. 始点s=a, aを選択: S={a}



2. aから出る辺に緩和操作

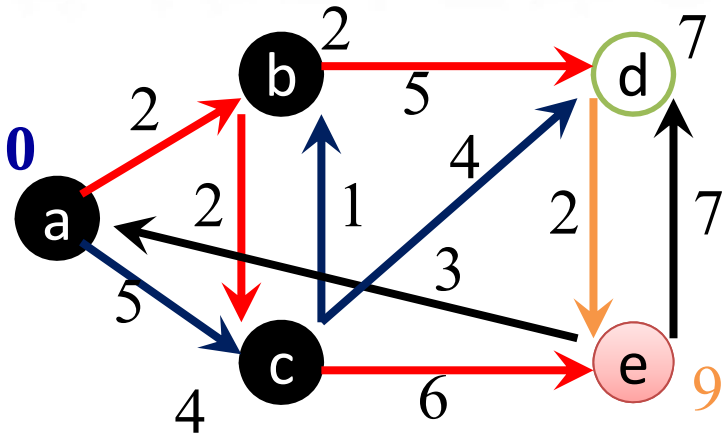


3. bを選択：S={a,b}.
bから出る辺に緩和操作

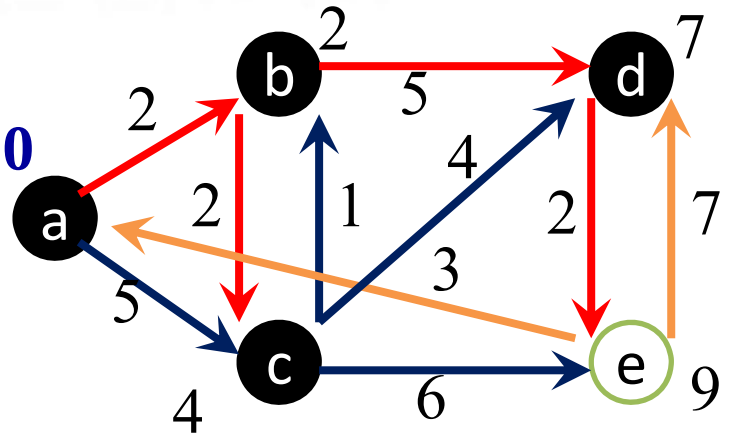


4. cを選択：S={a,b,c}.
cから出る辺に緩和操作

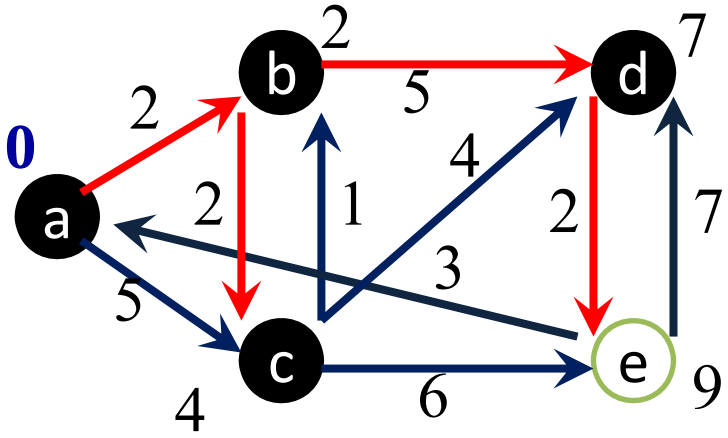
ダイクストラ法の実行例(2/2)



5. dを選択：S={a,b,c,d}.
dから出る辺に緩和操作



6. eを選択：S={a,b,c,d,e}.
eから出る辺に緩和操作



最終的な最短路木(赤の矢印)

ダイクストラ法の計算時間:
単純なデータ構造で V–S を管理

```
for v∈V do d[v]=∞; p(v) = nil; end        O(|V|)
d[s]= 0; S=∅;
while(V–S ≠ ∅) do
                                          O(|V–S|)
                                          O(|S|)
                                          O(|E_u|)
end                                       |V|
```

$$O\left(|E| + \sum_{S=\{s\}}^{V} |V - S|\right) = O(|E| + |V|^2)$$

ダイクストラ法の計算時間:
平衡2分探索木でV–Sを管理

キーはd[x]

```
for v∈V do d[v]=∞; p(v) = nil; end        O(|V|)
d[s]= 0; S=∅;
while(V–S ≠ ∅) do
                                          O(log |V|)
                                          O(log |V|)
                                          O(|E_u| log |V|)
end                                       |V|
```

d[x]が変化したら
VSも変化する

$$O\left(\sum_{u\in V}|E_u|\log|V|+\sum_{S=\{s\}}^{V}\log|V|\right)=O((|E|+|V|)\log|V|)$$

ダイクストラ法の計算時間: ヒープで V−S を管理

キーはd[x]

```
for v∈V do d[v]=∞; p(v) = nil; end        O(|V|)
d[s]= 0; S=∅;
while(V−S ≠ ∅) do
                                           O(1)
                                           O(log |V|)
                                           O(|E_u| log |V|)
end                                        |V|
```

d[x]が変化したら
VSも変化する

$$O\left(\sum_{u\in V}|E_u|\log|V|+\sum_{S=\{s\}}^{V}\log|V|\right)=O((|E|+|V|)\log|V|)$$

ダイクストラ法の計算時間: 比較

- V–Sを
  - 単純なデータ構造で管理: $O(|E| + |V|^2)$
  - 平衡二分探索木で管理: $O((|E| + |V|) \log |V|)$
- 従って、
  - $|E| \in \Omega(|V|^2)$なら単純なデータ構造を使うとよい
  - $|E| \in o(|V|^2)$なら平衡2分探索木を使うとよい

$f(n) \in o(g(n)) \Leftrightarrow \lim_{n\to\infty} |f(n)/g(n)| = 0$

ダイクストラ法の高速化: フィボナッチヒープを使う

- フィボナッチヒープでは
  - 最小値の発見: O(1)
  - 値の挿入: O(1)
  - (最小値を含む)値の削除: O(log n)
- ダイクストラ法の計算時間は全体として

$$O(|E| + |V| \log |V|)$$

ミニ演習

- スモールオー記法の練習
  - 定義: $f(n) \in o(g(n)) \iff \lim_{n \to \infty} \left| \frac{f(n)}{g(n)} \right| = 0$
- 以下，すべて違う関数をあげよ
  - $o(n^2)$ となる n の関数
  - $o(n)$ となる n の関数
  - $o(\log n)$ となる n の関数
  - $o(1)$ となる n の関数