KYOTO UNIVERSITY

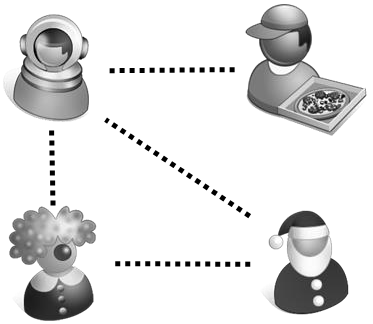
アルゴリズムとデータ構造⑨
～ グラフ ～

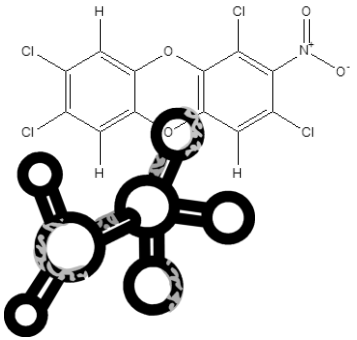
鹿島久嗣

DEPARTMENT OF INTELLIGENCE SCIENCE AND TECHNOLOGY

グラフ：
定義と基本アルゴリズム

- グラフの定義
  - –グラフ、木、配列
- グラフ上のアルゴリズム
  - –探索
  - –最短経路
  - –最大流（次回）



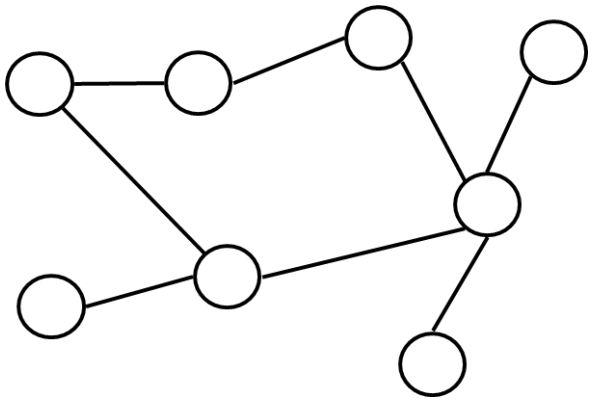


グラフ

グラフ：
頂点を辺でつないだもの

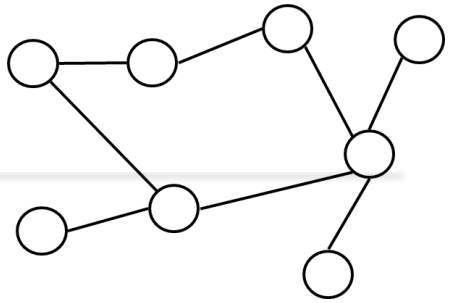
- グラフ$G = (V, E)$：頂点を辺（＝枝）でつないだもの
  - V：頂点集合（有限集合）
  - E：辺の集合（V 上の2項関係；$E \subseteq V \times V$）

直積集合



- 辺$e = (u, v) \in E$に向きがあるかどうかで有向グラフ、無向グラフに分類される
  - u, v は「隣接する」という
- グラフの例：交通網、Web、ソーシャルネットワーク、生体ネットワーク、化合物、構文解析木、…

グラフ関連の用語定義①：部分グラフ、パス



- 部分グラフ：
  2つのグラフ $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$が
  $V_1 \subseteq V_2$かつ$E_1 \subseteq E_2$のとき G_1はG_2の部分グラフであるという
- パス（道）：頂点系列$v_1, v_2, \ldots, v_k$で、
  $(v_1, v_2) \in E$ （長さ$k-1$）であるもの
  - $v_1, v_2, \ldots, v_k$がすべて異なるときパスは単純であるという
  - $v_1, v_2, \ldots, v_{k-1}$がすべて異なり$v_1 = v_k$のとき閉路という
  - 有向グラフのとき：パス（含まれる辺の向きが揃ってなくともよい）と有向パス（向きが揃っている）がある

グラフ関連の用語定義②：連結、木

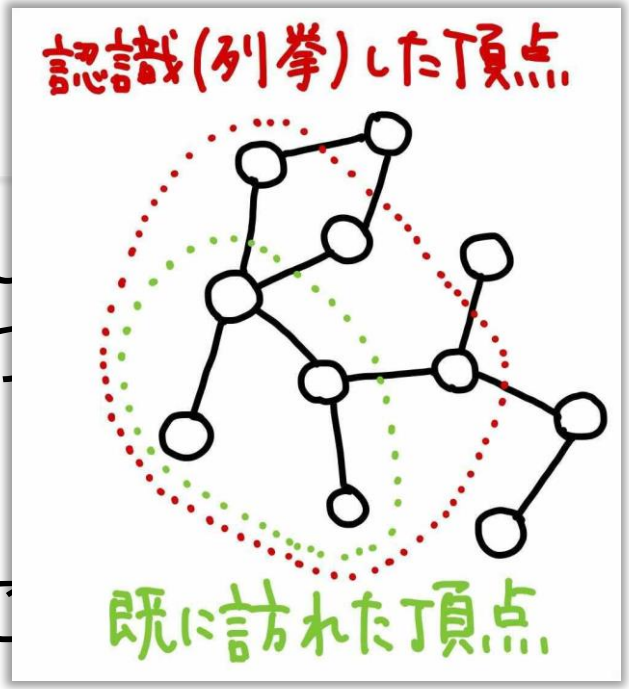
- 連結：G上のどの2点(u, v)に対しても、uからvへの（有向）パスがあるとき、Gは連結であるという
  - 特に、有向グラフの場合、uからvへのパスと、vからuへパスの両方があるとき、強連結という
- 木：閉路のない、連結な無向グラフ
  - 根付き木：根とよばれる特別な頂点をもつ木
  - 頂点数をnとすると辺の数は$n - 1$本

グラフ上の探索

グラフ上の探索：グラフの頂点列挙問題

- Webのクローリング：あるページから出発して、リンクをたどりながらページを列挙する（Webの地図をつくる）
- G上のある頂点v_0から開始して、G上を巡回してすべての頂点を列挙することを考える
  - 仮定：既に訪れた頂点に隣接する頂点は認識できる（列挙したことにできる）
- 基本方針：これまでに挙げた頂点に隣接する頂点のうち、まだ訪問していないものひとつを選んで移動…を繰り返す

グラフ上の探索：グラフの頂点列挙問題



- Webのクローリング：あるページから出発し
  ながらページを列挙する（Webの地図を
- G上のある頂点v_0から開始して、
  G上を巡回してすべての頂点を列挙するこ
  - 仮定：既に訪れた頂点に隣接する頂点は認識できる
    （列挙したことにできる）
- 基本方針：これまでに認識（列挙）した頂点のうち、
  まだ訪問していないものひとつを選んで移動…を繰り返す

グラフ上の頂点列挙の方針：列挙済み／未訪問の頂点の管理が肝

- 考えるべき２つのデータ構造：

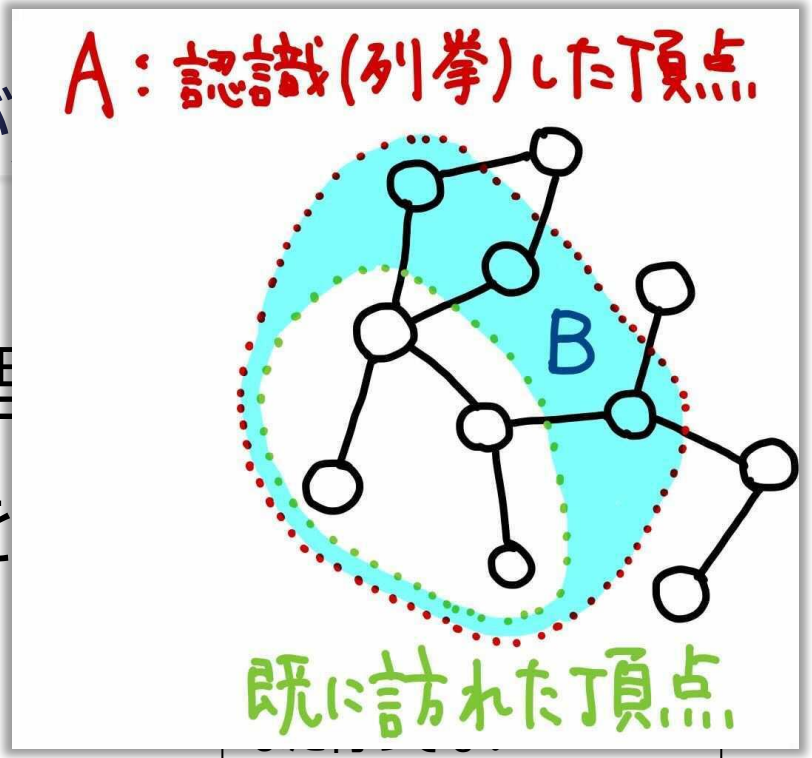
  A：すでに列挙した頂点集合を管理するデータ構造

  B：これから訪問すべき頂点集合を管理するデータ構造

1. v_0をAとBにいれる
2. Bから頂点をひとつ（v）とりだす
3. vに隣接する頂点でAに入っていないものがあれば、それらを全てAとBの両方に加える
4. Bが空なら、Aがすべての頂点集合。そうでなければ2へ。

認識（列挙）はしたがまだ行ってない

グラフ上の頂点列挙の方針：列挙済み／未訪問の頂点の管理が



- 考えるべき２つのデータ構造：

  A：すでに列挙した頂点集合を管理

  B：これから訪問すべき頂点集合を

1. v_0をAとBにいれる
2. Bから頂点をひとつ（v）とりだす
3. vに隣接する頂点でAに入っていないものがあれば、それらを全てAとBの両方に加える
4. Bが空なら、Aがすべての頂点集合。そうでなければ2へ。

列挙済み頂点の管理：ハッシュを使えば効率的にできる

- 考えるべき２つのデータ構造：

  A：**すでに列挙した頂点集合の管理**

  B：これから訪問すべき頂点集合の管理

- ある頂点を既に列挙したかどうかを効率よくチェックする：vに隣接する頂点集合$N(v)$ のそれぞれがAに含まれているか？
- 素朴にやると： $\mathrm{O}(|A||N(v)|)$
- ハッシュを使って：$\mathrm{O}(|N(v)|)$

これから訪問すべき頂点の管理：キューやスタックで管理する

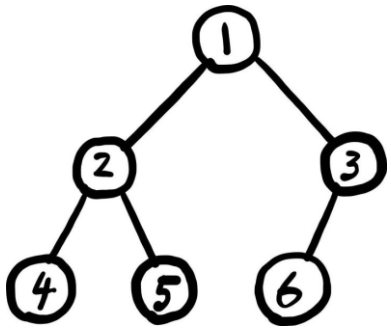
- 考えるべき２つのデータ構造：

  A：すでに列挙した頂点集合の管理

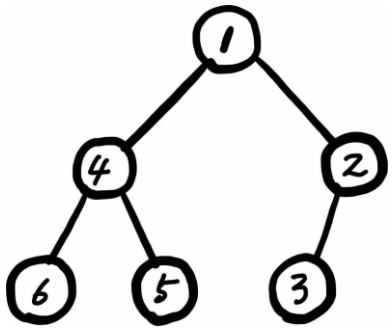
  B：これから訪問すべき頂点集合の管理

- 2通りの実現法：実現方法によって訪問順が変わる
  - キュー：幅優先探索
  - スタック：深さ優先探索

幅優先探索

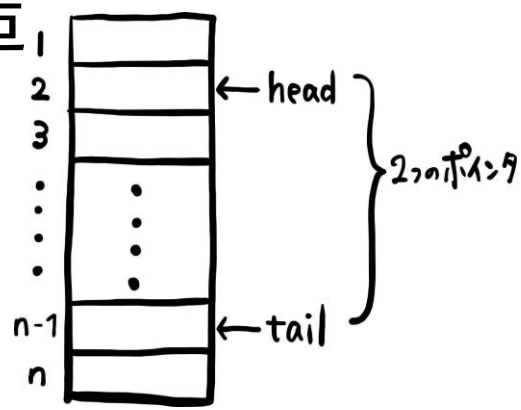


深さ優先探索

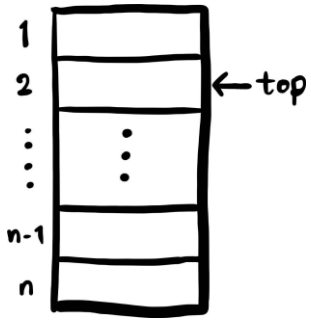


キューとスタック：
それぞれ FIFO / LIFO のデータ構造

- キュー：First-in-first-out (FIFO)のデータ構造
  - ２つのポインタ：headとtail
  - 追加：tail位置に追加して、tail+1
  - 取り出し：head位置を読み出して、head+1



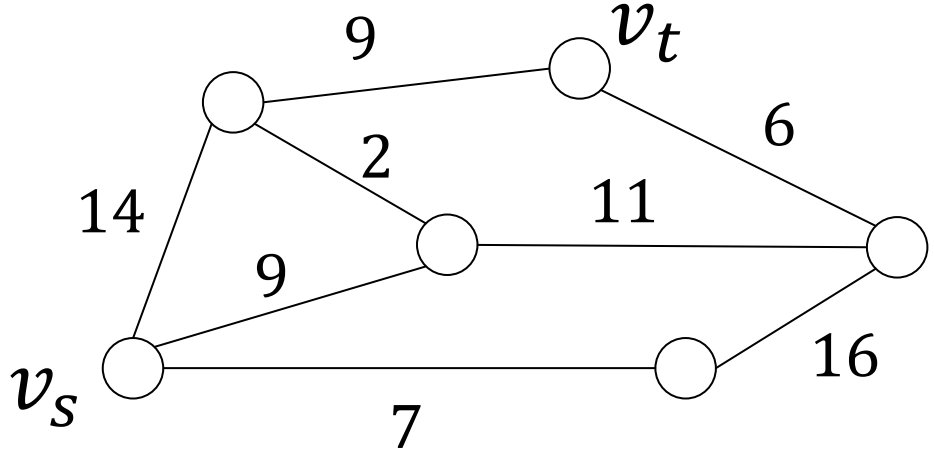
- スタック：Last-in-first-out (LIFO)のデータ構造
  - ポインタ：top
  - 追加：top+1の位置に追加して、topを+1する
  - 取り出し：top位置を取り出し、topを-1する



最短経路問題

グラフ上の最短経路問題：始点から終点へのコスト最小のパスをみつける問題

- グラフ$G = (V, E)$において：
  - 各辺$e \in E$に、非負の実数コスト$l(e)$が与えられている
  - 特別な頂点である始点v_sと終点v_tがある
- 始点v_sから終点v_tへ至るパスのうち、パス上の枝のコストの和が最小になるようなパスをみつけたい



ダイクストラ法：始点から各頂点への最短コストの更新を繰り返す

- $d(v)$を（これまでに見つかった）始点からvまでのパスの最小コスト

1. $d(v_s) \leftarrow 0;\ V - \{v_s\}$に属するすべての頂点 vに対して$d(v) \leftarrow \infty;\ A \leftarrow \{v_s\}$
2. Aの頂点のうち、$d(v)$が最小のvをとりだす
3. $v = v_t$ ならば、$d(v_t)$を出力して終了
4. vに隣接する各頂点wに対して：
   1. if $d(w) = \infty$ then wをAに追加して $d(w) \leftarrow d(v) + l(v, w)$
   2. else if $d(w) > d(v) + l(v, w)$ then $d(w) \leftarrow d(v) + l(v, w)$
5. Step 2 にもどる

Aはまだ最短コストが確定していない頂点集合

初めて訪問したケース

2回目以降の訪問ではより短い経路が見つかれば置き換える

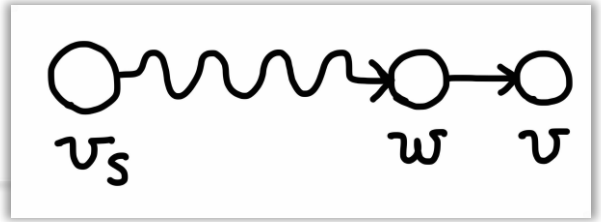
ダイクストラ法の計算量：ヒープを使って実現すると$O(|E|\log|V|)$でできる

- 初期化の計算量は$O(|V|)$：ノード数$|V|$に比例
- Aの頂点のうち$d(v)$が最小のvをとりだす計算量は$O(\log|V|)$
  - $d(v)$をプライオリティキュー（ヒープ等）で管理するとする
  - 全部まとめて$O(|V|\log|V|)$
- vに隣接する各頂点wをチェックする計算量は：
  - vに隣接する頂点数$|N(v)|$
  - $w \in N(v)$, $d(w)$の更新にあわせ、ヒープを作り変えるのは$O(\log|V|)$
  - $\sum_{v\in V}|N(v)| = 2|E|$ なので、全部まとめると $O(|E|\log|V|)$

ダイクストラ法の正当性：このアルゴリズムは最短経路を求めることが保証される

- 定理：ステップ２「Aの頂点のうち、$d(v)$が最小のvをとりだす」で、Aの頂点のうち$d(v)$が最小の頂点vは、その時点の$d(v)$がv_Sからvまでの最短経路長になっている
  - つまりステップ２で頂点が取り出されるたびに、頂点がひとつずつ最短経路が確定していく
- 数学的帰納法による証明：
  I. ステップ２が初めて実行されたとき、$d(v) = d(v_S) = 0$ である
  II. ある時点まで定理が成り立っていると仮定すると、
  III. 次にステップ２が実行されるときに定理が成り立つことを示す

ダイクストラ法の正当性：数学的帰納法による証明（III）



III. 次にステップ２が実行されるときに成り立つことを示す

- とりだされたvまでの最短経路においてvのひとつまえの頂点をwとする
- wが過去に取り出された頂点集合Fに含まれているかで場合わけ：
  - 含まれている場合：$d(w)$は最短経路長。以前wをとりだしたときのステップ４で$d(v)$はすでに最短経路長に更新されているはず
  - 含まれていない場合：
    - 経路上で初めてFに含まれていない頂点xに対し、そのひとつ前のyに対する$d(y)$は最短経路長
    - 以前yをとりだしたときのステップ４で$d(x)$はすでに最短経路長に更新されているはず。それならばvよりもxのほうが先に取り出されるべき

      → （実際にvが取り出されているということは）そのようなxはない

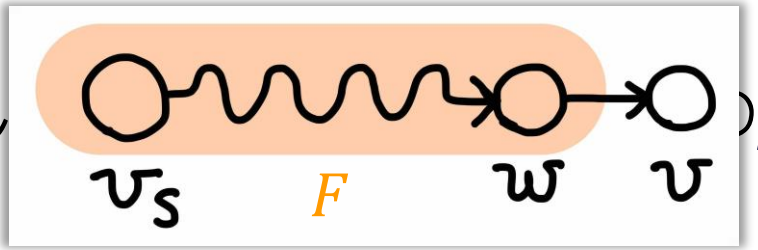
ダイクストラ法の正当性：数学的帰納法による証明（III）

III. 次にステップ２が実行されるときに成り立つことを示す

– とりだされたvまでの最短経路においてvのひとつまえの頂点をwとする

– wが過去に取り出された頂点集合Fに含まれているかで場合わけ：

- 含まれている場合：$d(w)$は最短経路長。以前wをとりだしたときのステップ４で$d(v)$はすでに最短経路長に更新されているはず
- 含まれていない場合：
  – 経路上で初めてFに含まれていないyに対する$d(y)$は最短経路長
  – 以前yをとりだしたときのステップ４で$d(x)$はすでに最短経路長に更新されているはず。それならばvよりもxのほうが先に取り出されるべき
    → （実際にvが取り出されているということは）そのようなxはない



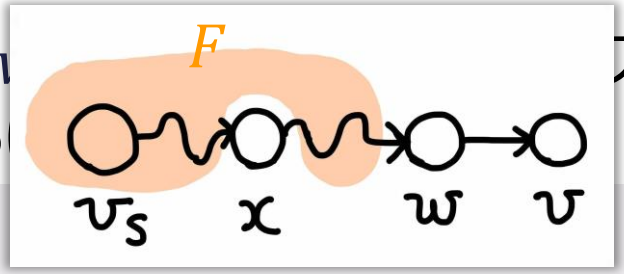
ダイクストラ法の正当性：数学的帰納法による証明（III）

III. 次にステップ２が実行されるときに成り立つことを示す

– とりだされたvまでの最短経路においてvのひとつまえの頂点をwとする

– wが過去に取り出された頂点集合Fに含まれているかで場合わけ：

- 含まれている場合：$d(w)$は最短経路長。以前wをとりだしたときのステップ４で$d(v)$はすでに最短経路長に更新されている

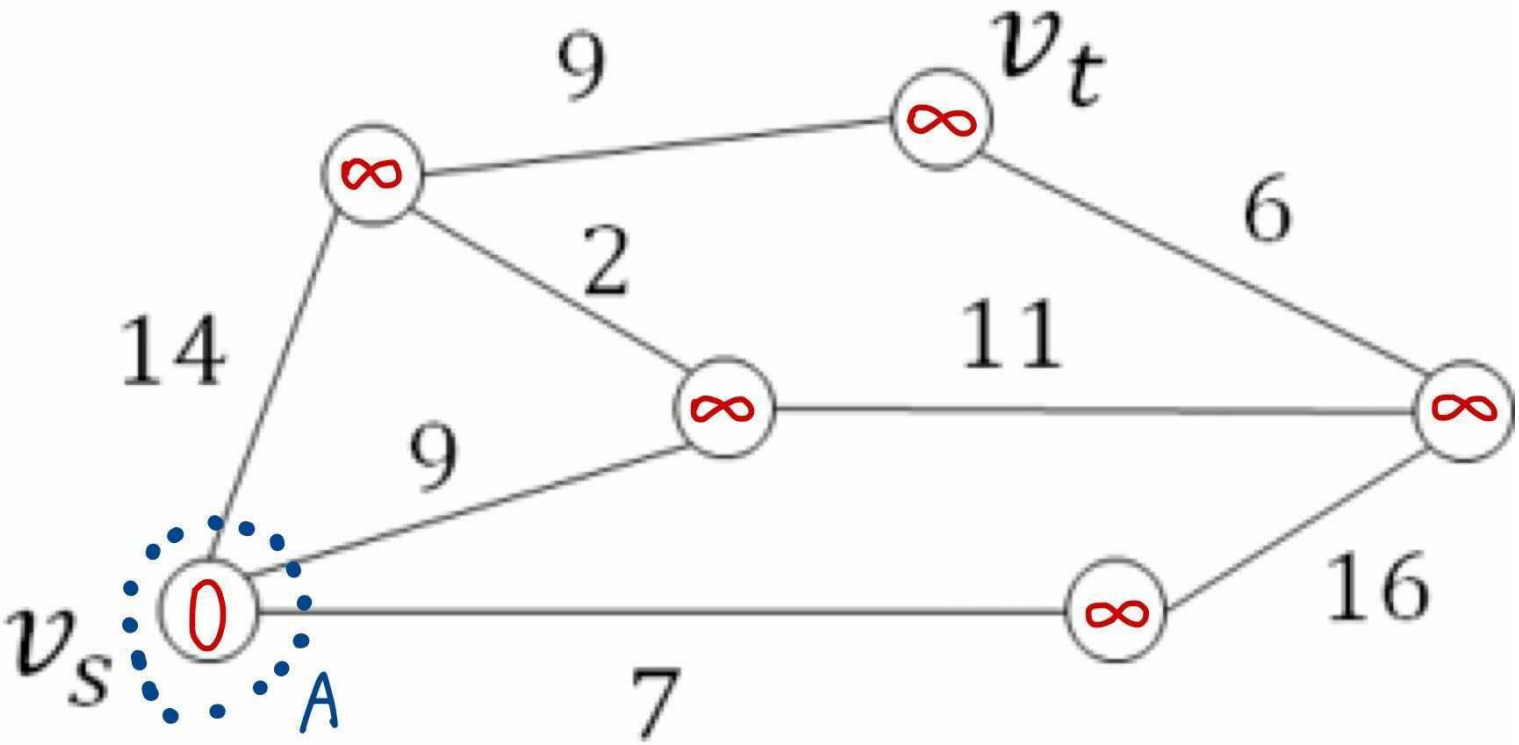


- 含まれていない場合：
  – 経路上で初めてFに含まれていない頂点xに対し、そのひとつ前のyに対する$d(y)$は最短経路長
  – 以前yをとりだしたときのステップ４で$d(x)$はすでに最短経路長に更新されているはず。それならばvよりもxのほうが先に取り出されるべき
    → （実際にvが取り出されているということは）そのようなxはない

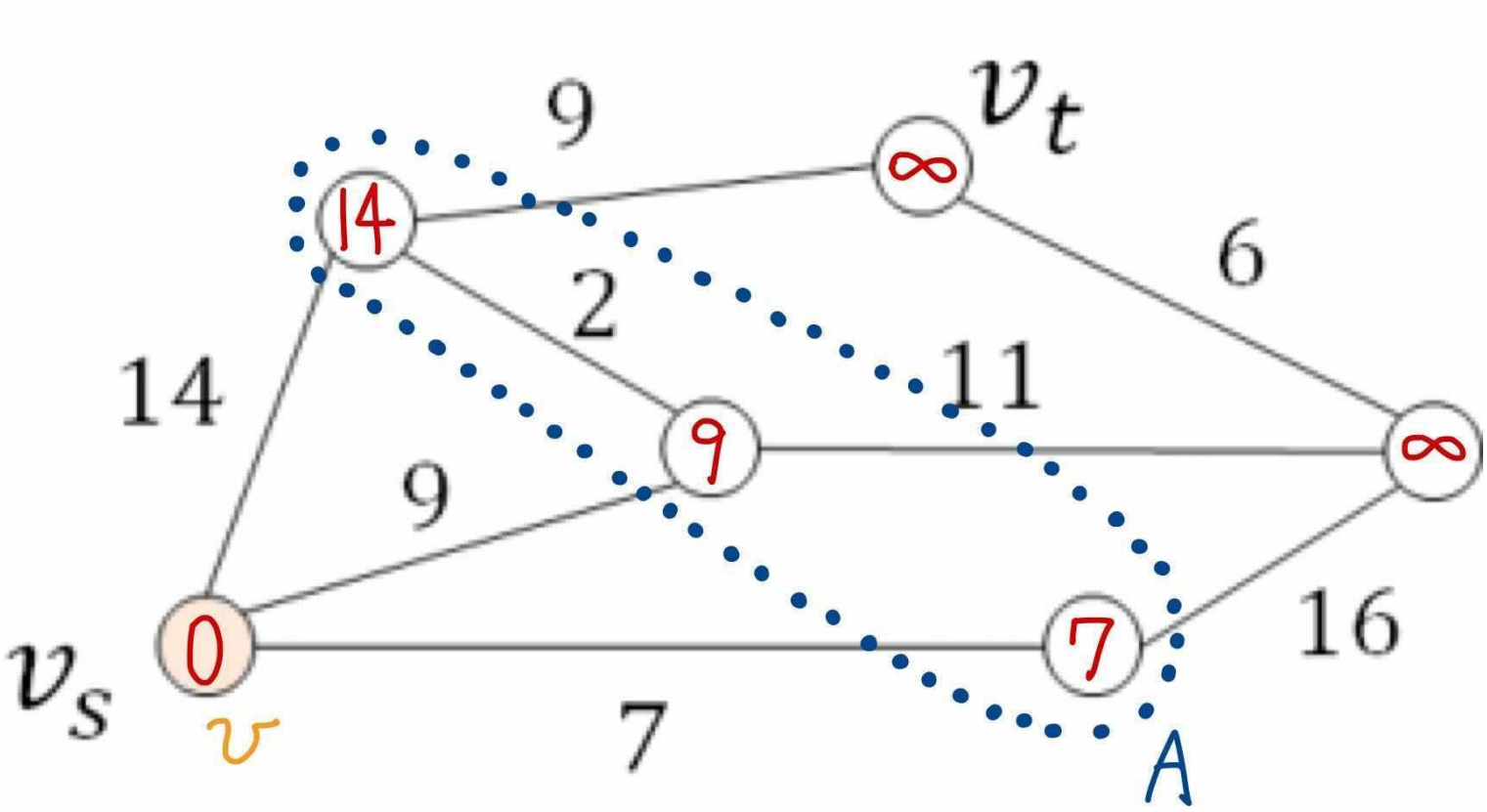
A*アルゴリズム：最短経路の見積もりを使って探索を効率化

- ダイクストラ法では$d(v)$として現時点まで明らかになっている最短経路長の上界をつかって探索順序を決定した
  - つまり、未来の情報はつかっていない
- $d(v)$を一般化する：$a(v) = d(v) + h(v)$
  - ヒューリスティクス関数$h(v)$： vからv_tへの最短経路の下界
    - たとえば2次元平面上での問題であれば、vとv_tのユークリッド距離を使用できる
      - ゴールを向いた方向を優先して探索する
    - ダイクストラ法では$h(v) = 0$

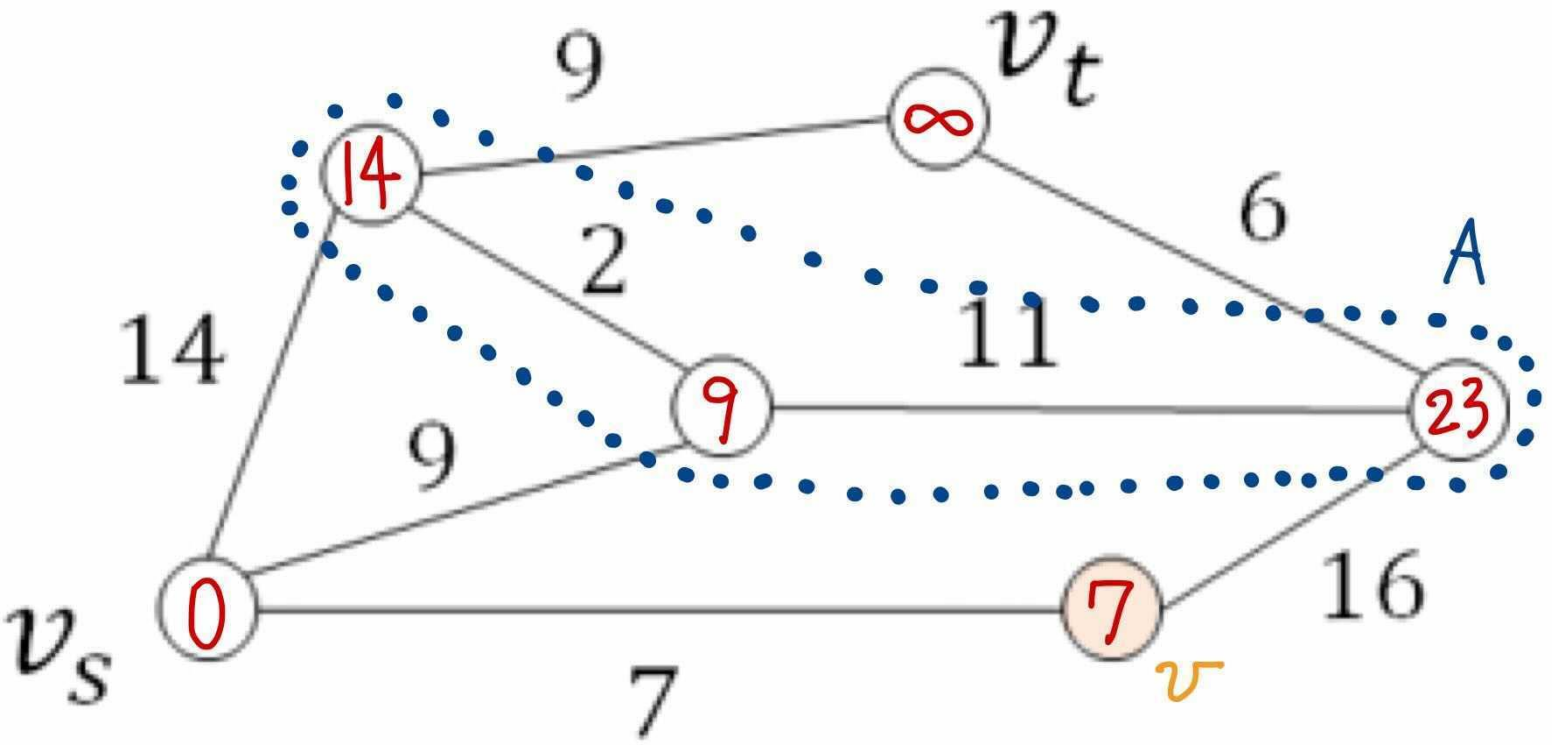
付録：
ダイクストラ法の例 – Step 1



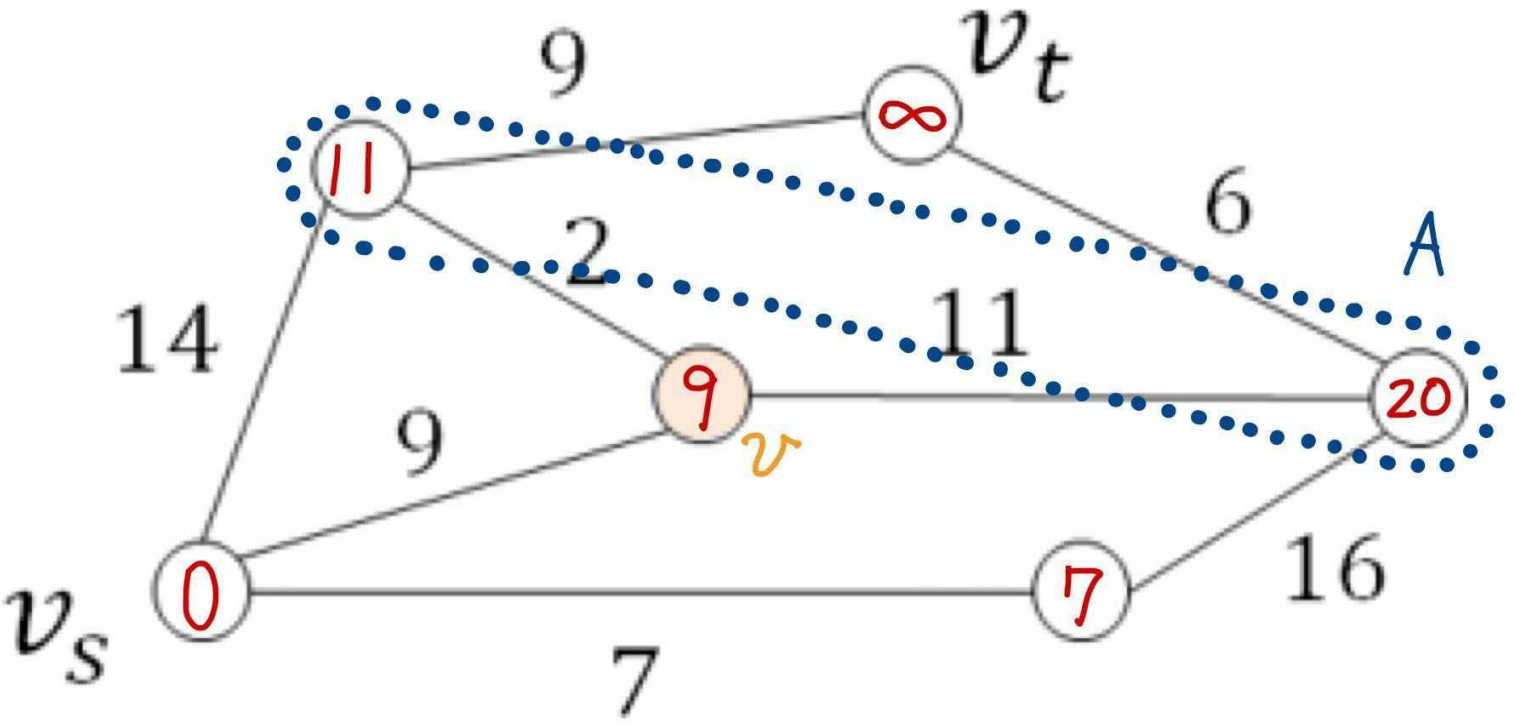
付録：
ダイクストラ法の例 – Step 2



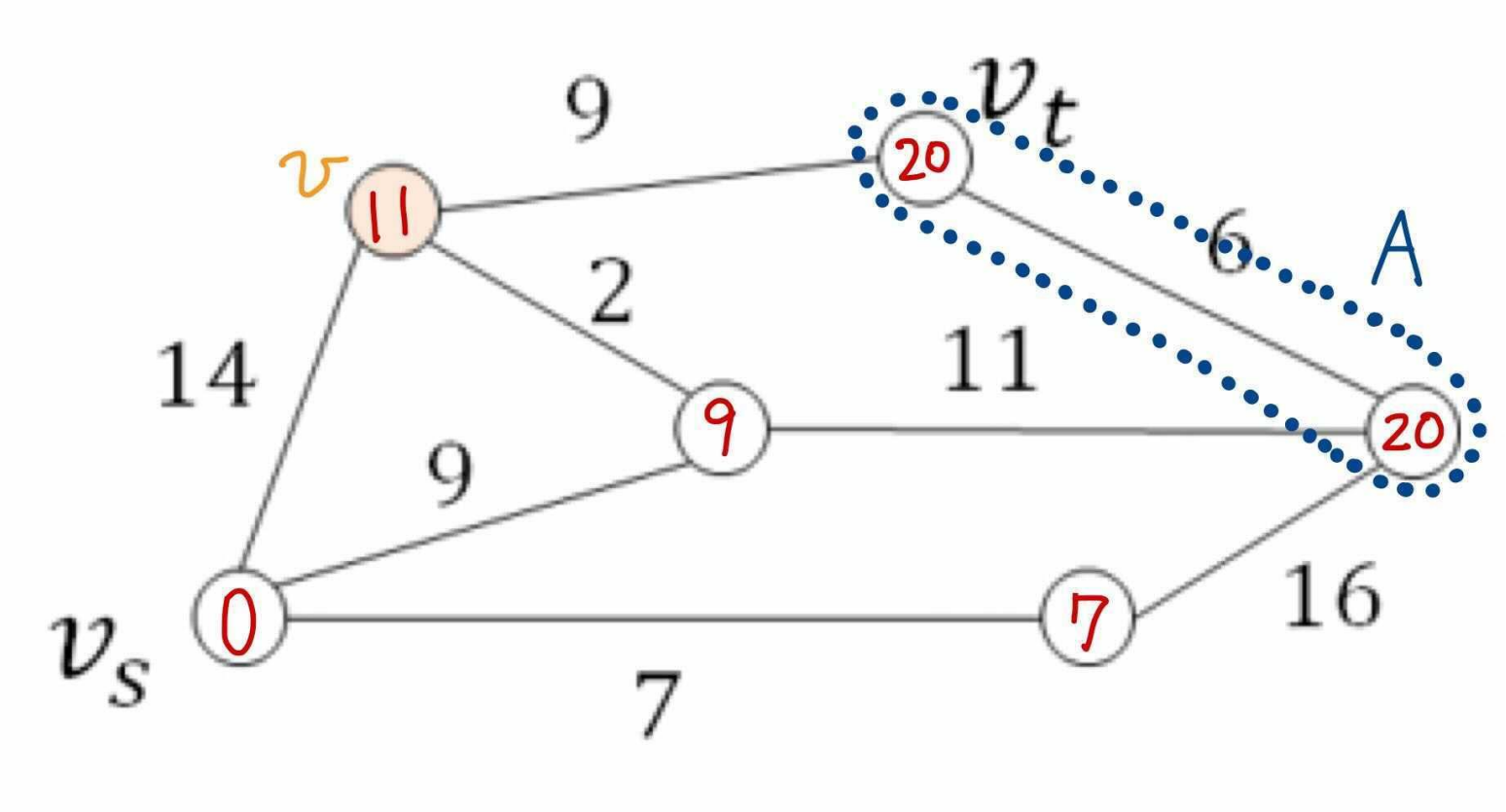
付録：
ダイクストラ法の例 – Step 3



付録：
ダイクストラ法の例 – Step 4



付録：
ダイクストラ法の例 – Step 5



付録：
ダイクストラ法の例 – Step 6