

アルゴリズムとデータ構造⑧

～ グラフ ～

鹿島久嗣

グラフ： 定義と基本アルゴリズム

■ グラフの定義

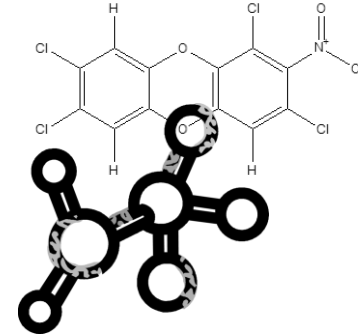
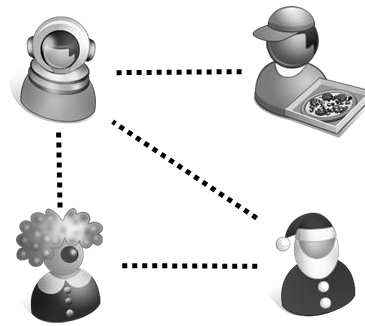
— グラフ、木、配列

■ 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

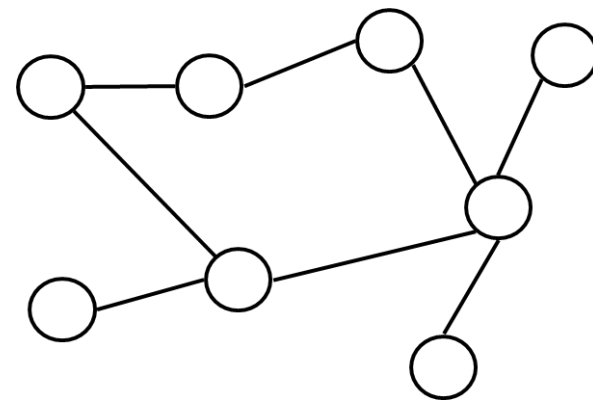
- グラフ $G = (V, E)$: 頂点を辺 (= 枝) でつないだもの

- 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

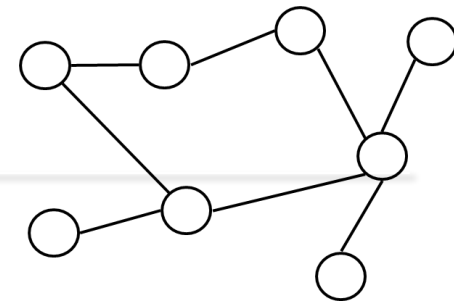
- 辺 $e = (u, v) \in E$ に向きがあるかどうかで
有向グラフ、無向グラフに分類される

- 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, \dots, v_k で、
 $(v_1, v_2) \in E$ （長さ $k - 1$ ）であるもの
 - v_1, v_2, \dots, 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, \dots, 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

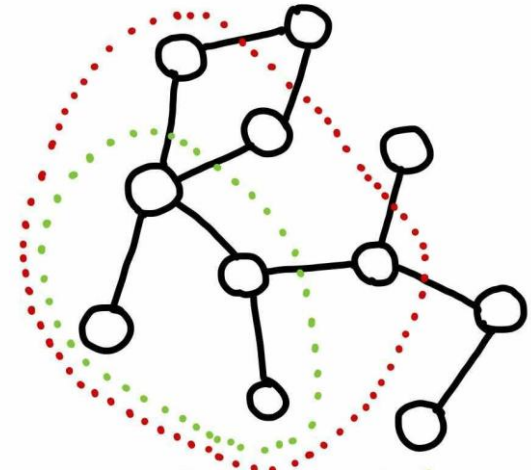
グラフ上の探索： グラフの頂点列挙問題

- 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（列挙したことによる）
- 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

グラフ上の探索： グラフの頂点列挙問題

- Webのクロール：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と
- 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

認識(列挙)した頂点



既に訪れた頂点

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

認識（列挙）はしたが
まだ訪問していない頂点

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。

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

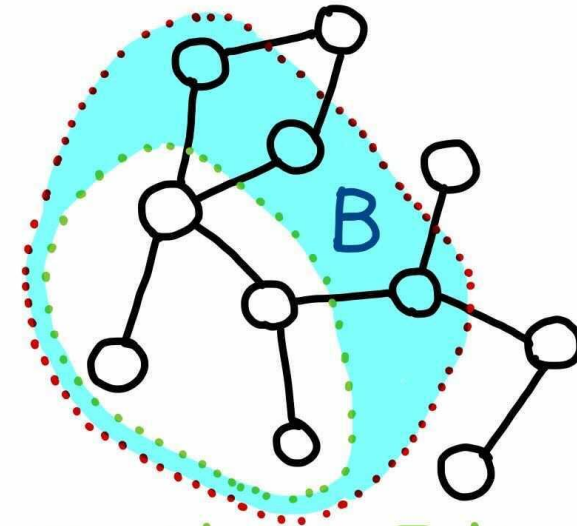
■考えるべき2つのデータ構造：

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。

A：認識(列挙)した頂点



既に訪れた頂点

列挙済み頂点集合 (A) 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 (A の中に入ったことがあるか) 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 : $O(|A||N(v)|)$
- ハッシュを使って : $O(|N(v)|)$

これから訪問すべき頂点集合 (B) 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

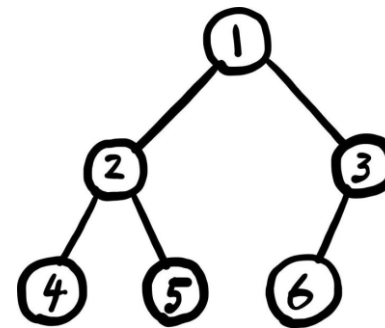
■ B の 2 通りの実現法 : 実現方法によって訪問順が変わる

— キュー : 幅優先探索

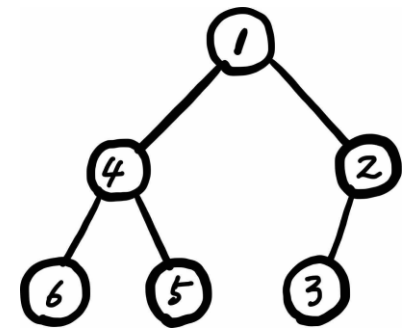
— スタック : 深さ優先探索

用いるデータ構造と探索順が
対応

幅優先探索



深さ優先探索

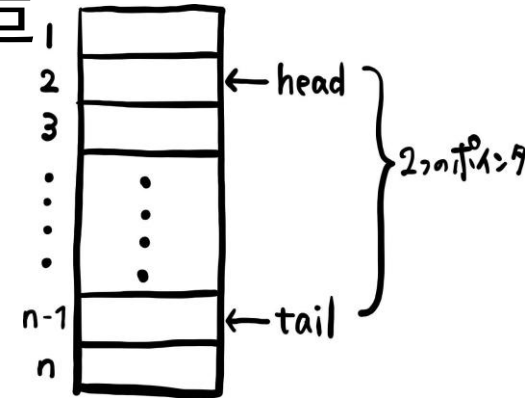


キューとスタック :

それぞれ FIFO / LIFO のデータ構造

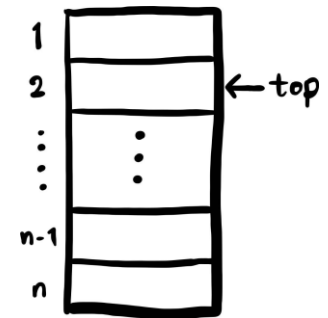
■ キュー : First-in-first-out (FIFO) のデータ構造

- 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

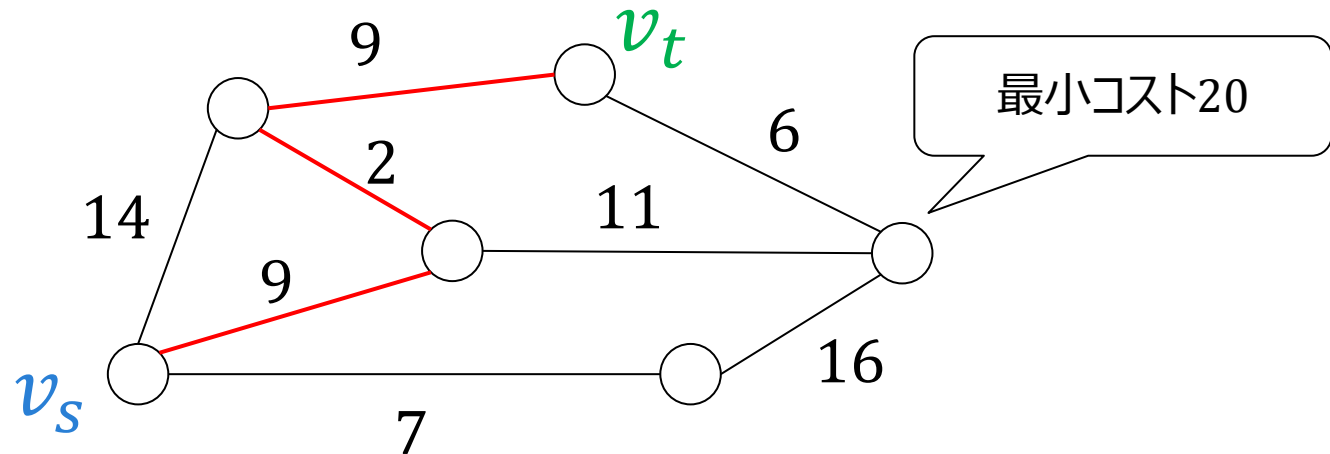


最短経路問題

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見つける問題

- 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（最短経路）をみつきたい



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

- $d(v)$ を始点から v までのパスの暫定最小コストとする

1. $d(v_s) \leftarrow 0$; $V - \{v_s\}$ に属するすべての頂点 v に対し $d(v) \leftarrow \infty$;
 $Q \leftarrow \{v_s\}$: 初期化

2. Q から、最小の $d(v)$ をもつ頂点 v をとりだす

Q は、発見したが、まだ最短コストが確定していない頂点の集合

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：

初めて訪問したケース

1. if $d(w) = \infty$ then w を Q 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

2回目以降の訪問ではより短い経路が見つければ置き換える

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
- Q 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 $|N(v)|$
 - 各 $w \in N(v)$ に対して、 $d(w)$ の更新にあわせてヒープを作り変える計算量は $O(\log |V|)$
 - $\sum_{v \in V} |N(v)| = 2|E|$ 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「 Q の頂点のうち、 $d(v)$ が最小の v をとりだす」で、 Q 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 で頂点を取り出されるたびに、その最短経路が確定していく
- 数学的帰納法による証明：

これが最短経路コストなのは明らか

 - 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

- 今とりだされた v までの最短経路において、 v のひとつ前の頂点を 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： $d(v)$ は既に最短経路長。過去に w を取り出したときのステップ 4 の時点で $d(v)$ は既に最短経路長に更新されているはずだから
 - 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）

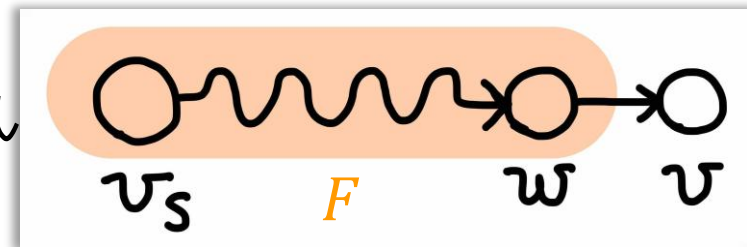
III. 次にステップ 2 が実行されるときに成り立つことを示す

- 今とりだされた v までの最短経路において、 v のひとつ前の頂点を 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： $d(v)$ は既に最短経路長。過去に w を取り出したときのステップ 4 の時点で $d(v)$ は既に最短経路長に更新されているはずだから

- 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 w に対して、 w に対する $d(w)$ は最短経路長



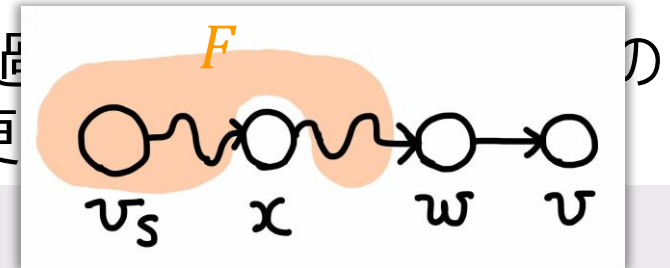
- 以前 y をとりだしたときのステップ 4 で $d(x)$ はすでに最短経路長に更新されているはず。それならば v よりも x のほうが先に取り出されるべき
→（実際に v が取り出されているということは）そのような x はない

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

- 今とりだされた v までの最短経路において、 v のひとつ前の頂点を 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： $d(v)$ は既に最短経路長。過去のステップ 4 の時点で $d(v)$ は既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

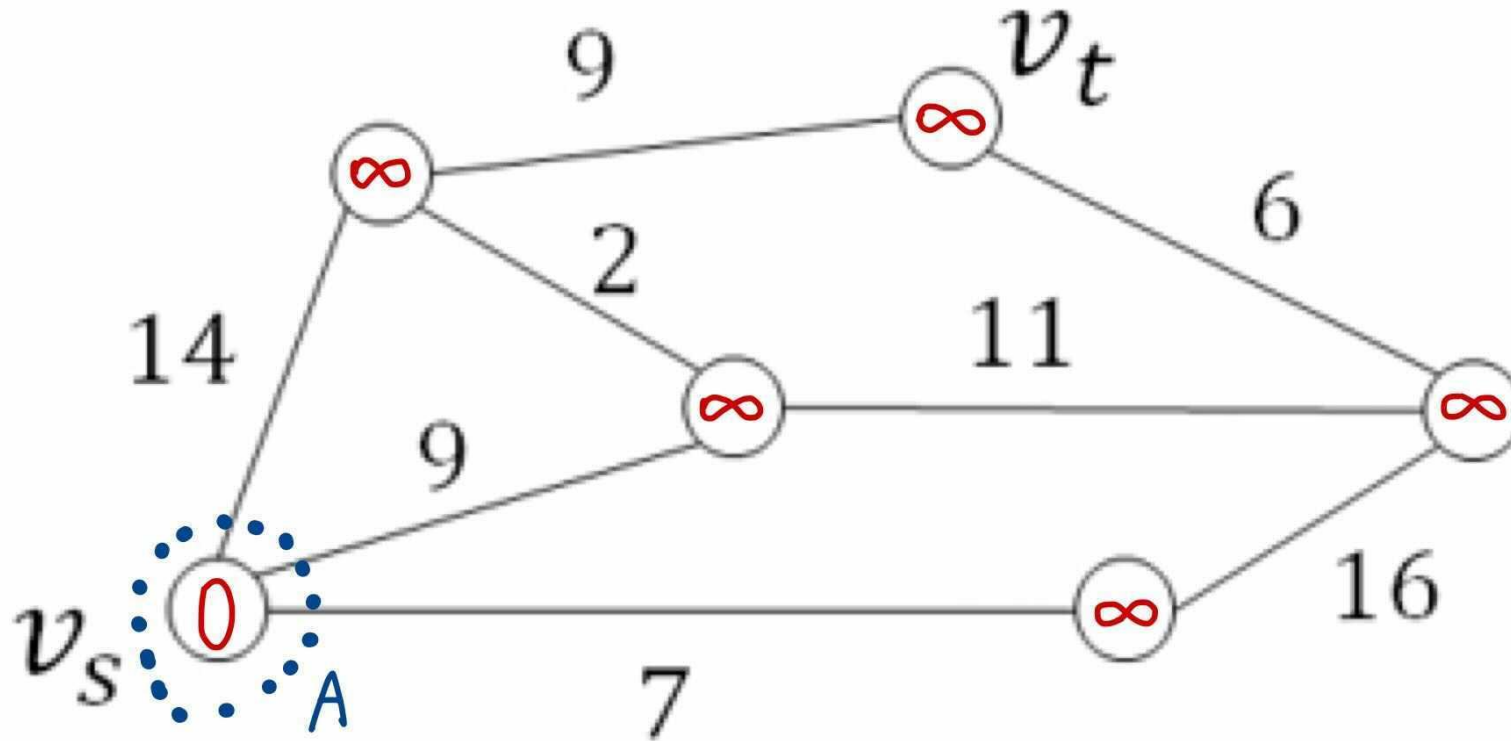


A*アルゴリズム：

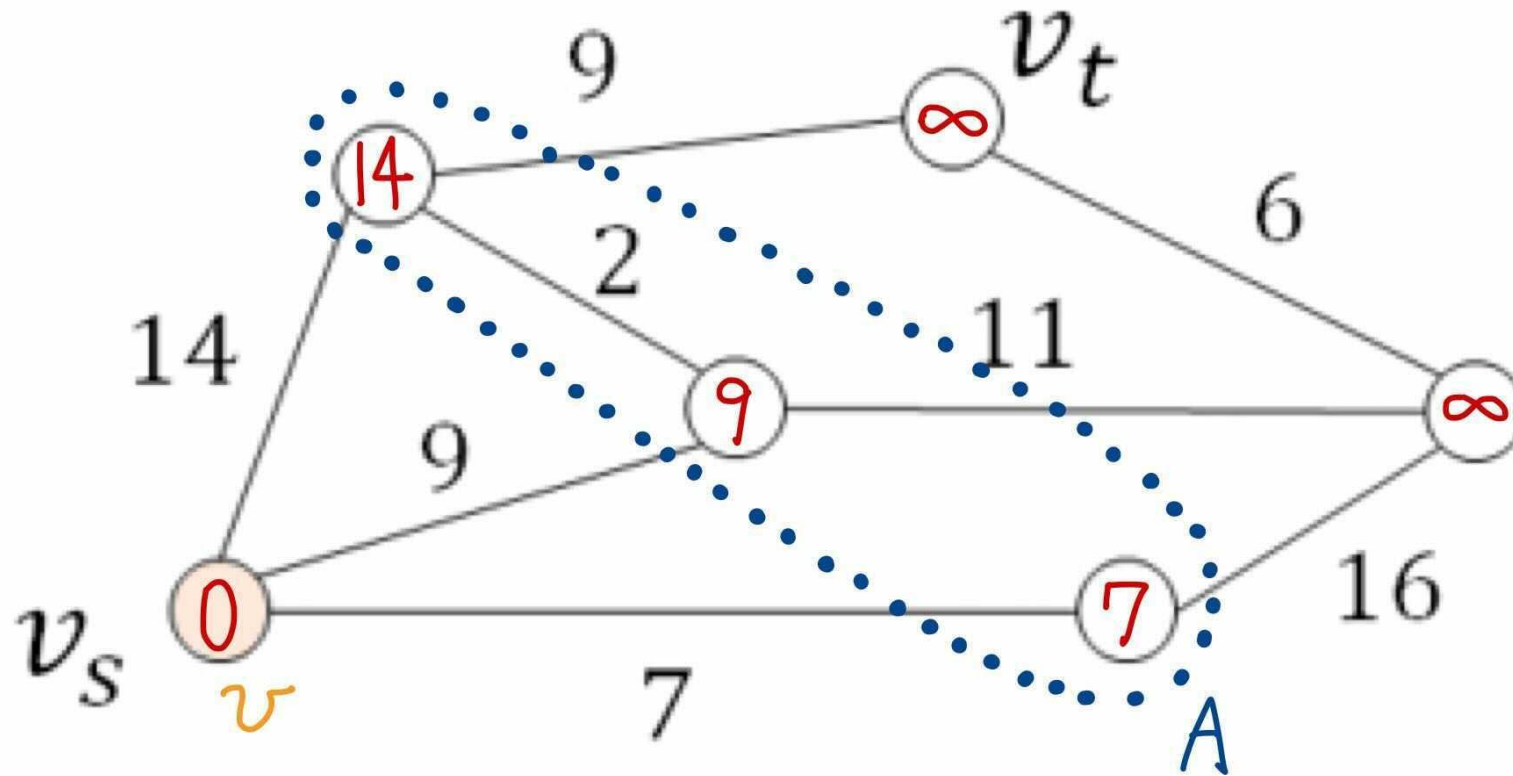
最短経路の見積もりを使って探索を効率化

- 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 $h(v) = 0$

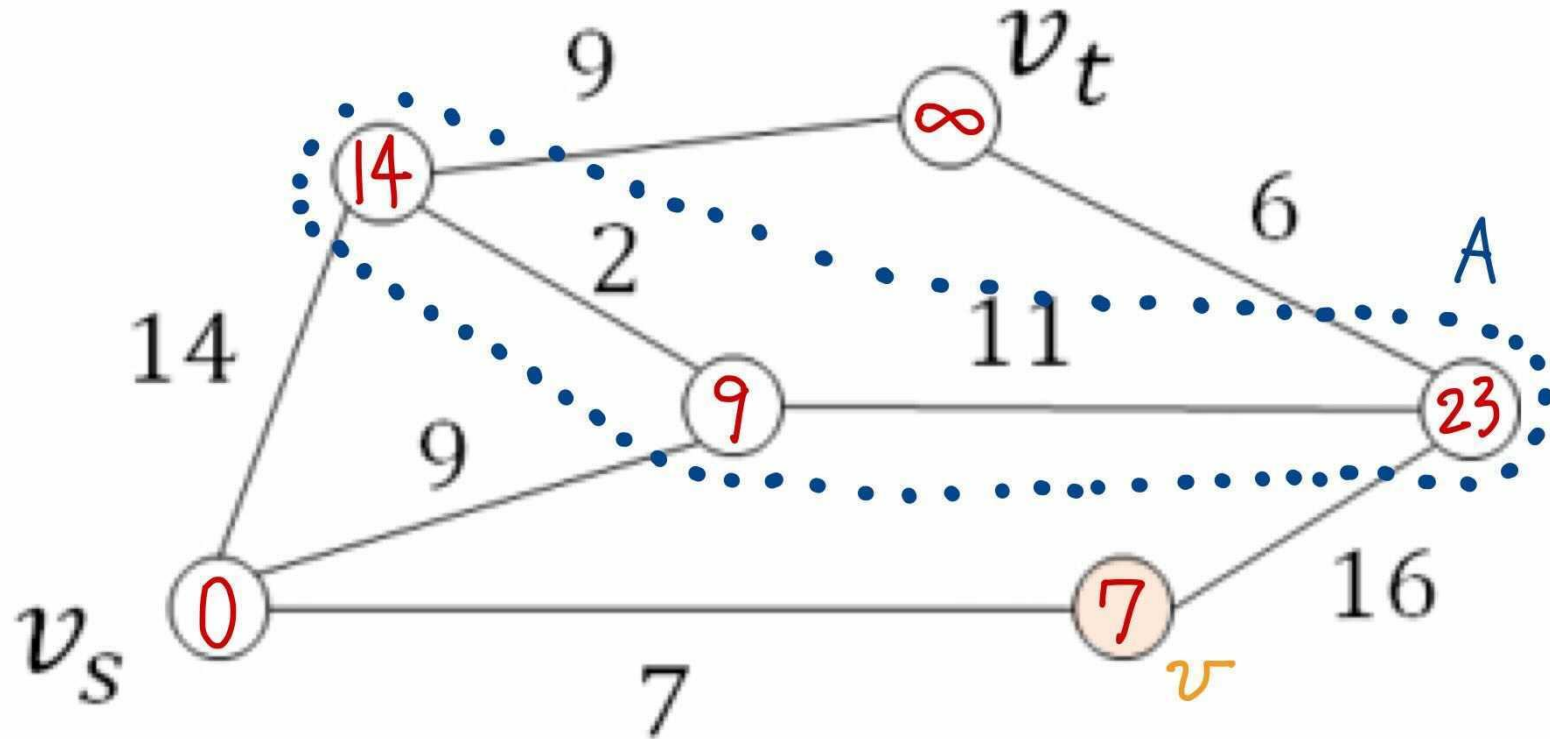
付録： ダイクストラ法の例 – Step 1



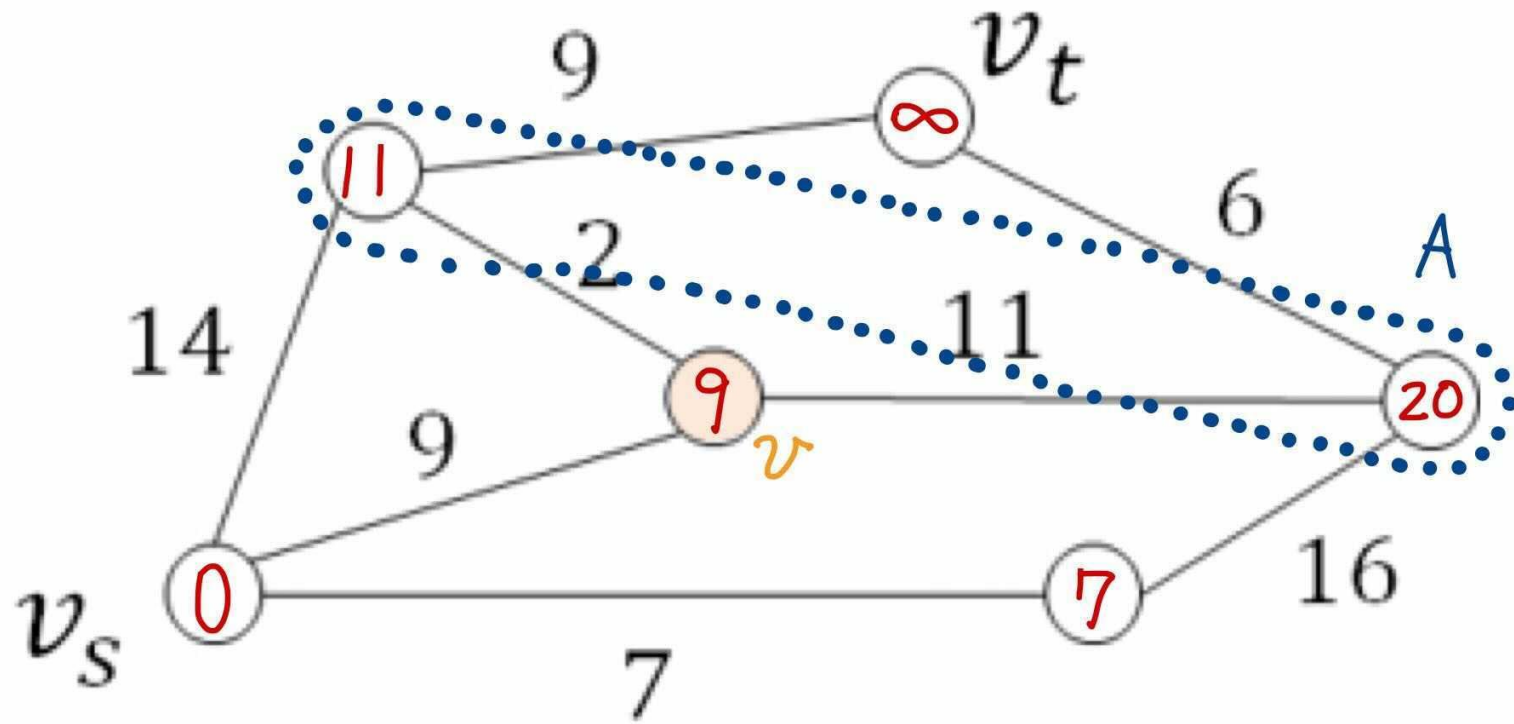
付録： ダイクストラ法の例 – Step 2



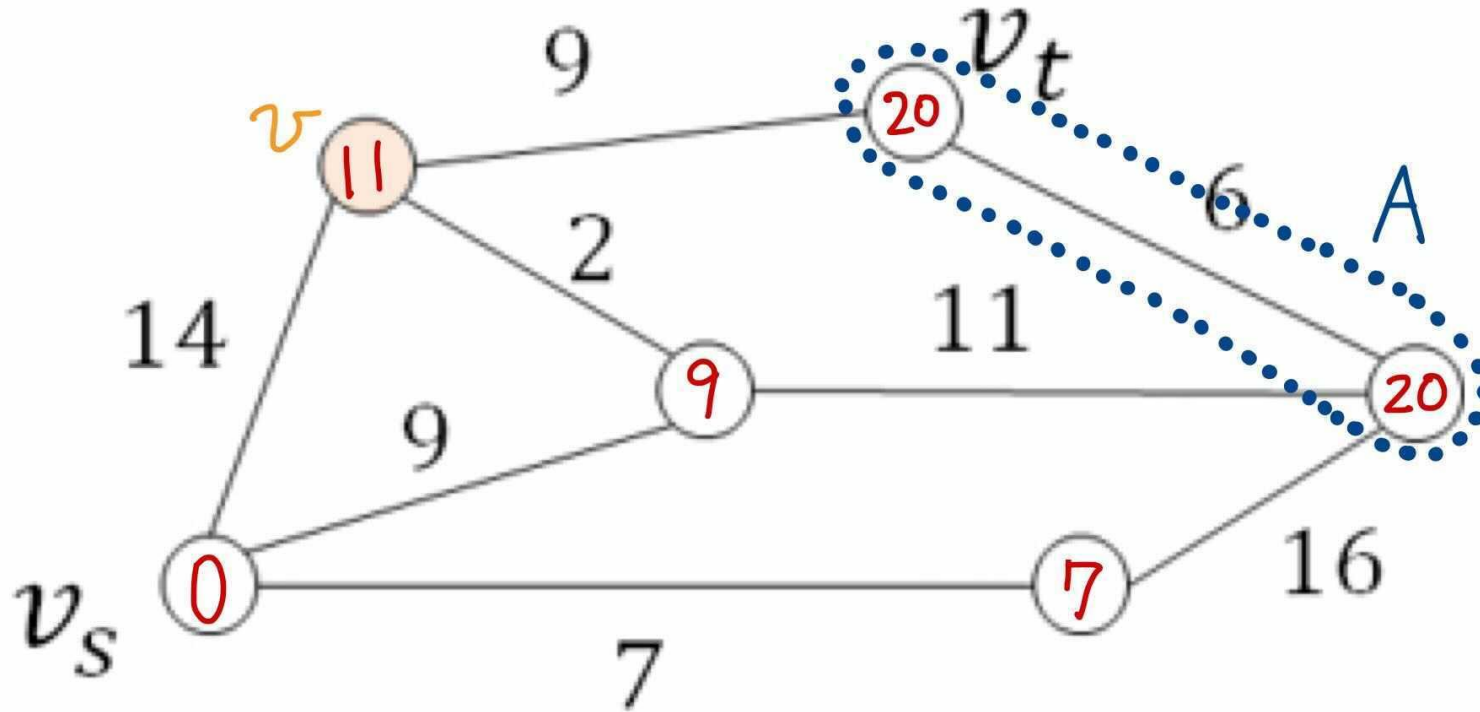
付録： ダイクストラ法の例 – Step 3



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

