講義２：最短経路問題２：

上原隆平
uehara@jaist.ac.jp

単一頂点対最短経路問題:

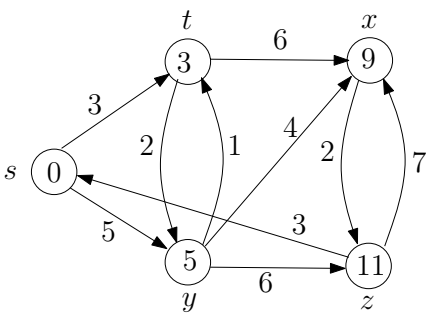
辺に正の重みをもつ無向グラフと，始点 s と目的地 t が指定されたとき，s から t までの重み最小の経路（最短経路）を求めよ.

最も重要な性質： 最短経路の部分構造最適性
頂点間の最短路は他の最短経路を内に含む.
つまり，最短経路の部分経路はまた最短経路である.

証明： 始点 s から目的地 t までの最短経路をPとし，その部分経路をQとする. もし，Qが最短経路でなければ，Pにおいて Qの部分経路を最短経路で置き換えればPより短い経路が得られることになるが，これはPが最短経路であることに矛盾する.

負の重みの辺の影響

負の重みをもつ辺が存在する場合には，最短経路問題は複雑になる．特に，重みの和が負であるような閉路が存在すると，その閉路を何度も回ると重みはどんどん減るのでいくらでも短い経路ができてしまう．



xからzへの辺xzを含む閉路は

xz, zx

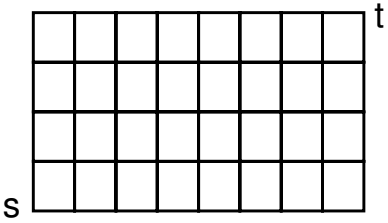
のみ．

辺xzの重みが負の値-2のとき，この閉路の重みの和は7-2>0なので問題は生じない．

しかし，xzの重みが-9になると，閉路の重みの和が負になるのでこの閉路を回る度に重みの和が減少する．

最短経路問題の難しさ

2点間の経路は非常に多数存在しうる.
簡単な例はマンハッタンでの経路選択:

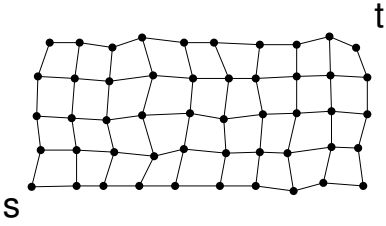


m本の縦の筋とn本の横の筋からなる街路において
左下の s から右上の t まで至る最短経路は${}_nC_{m+n}$通り.
これは m, n に関しては指数関数.

問題:マンハッタンの例で最短経路が${}_nC_{m+n}$通りあることを示せ.
m = n = 20のとき, この値はどの程度の大きさか?

最短経路問題の難しさ

２点間の経路は非常に多数存在しうる.
簡単な例はマンハッタンでの経路選択:



少し頂点が移動すると最短経路に近い経路が多数存在する.

m本の縦の筋とn本の横の筋からなる街路において
左下の s から右上の t まで至る最短経路は${}_nC_{m+n}$通り.
これは m, n に関しては指数関数.

したがって, 最短経路の候補をすべて列挙して最短のものを選ぶという素朴な方法では遅すぎる.

最短経路問題の難しさ

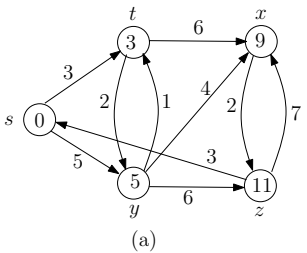
辺の重みとして負の値を許したとき，経路の重みの単調性（部分経路の長さは全体の経路より常に短いという性質）が成り立たないので，問題が難しくなる．

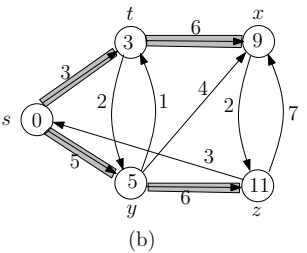
負の重みの閉路を許しても，同じ辺を2度通らない経路（単純な経路という）に限定すると，最短経路を定義することができる．しかし，この状況で単純な最短経路を求める問題は難しい（NP-困難）．

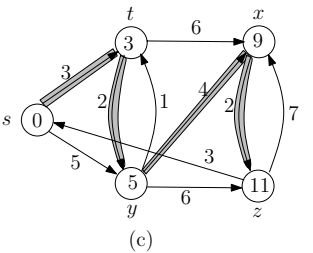
負の重みの閉路があっても，始点 s と目的地 t の間の経路と関係のないところにしか存在しなければ，最短経路を求めることができる．

グラフにおける最短経路の表現

始点sからグラフの各頂点までの最短経路は，それぞれの頂点で一つ前の頂点を記憶するだけで表現できる．



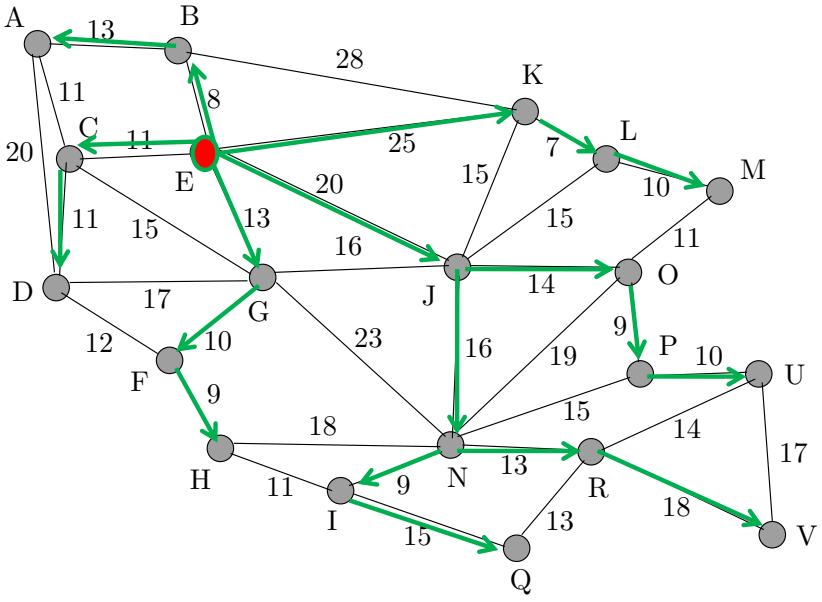




入力のグラフと
始点sからの距離

2通りの異なる最短路木
（どちらでも距離は同じ）

問題：もう一通り最短路木が存在する．それを求めよ．



E地点に関する最短路木.

最短路木

グラフGと，その一つの頂点sが与えられたとき，sに関するGの最短路木Tとは

- Tの頂点はGにおいてsから到達可能な頂点である.
- Tの根はsである.
- Tのすべての頂点 v について，s から v に至るTの経路はGにおける s から t への最短経路である.

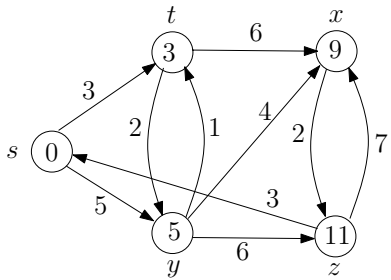
s から t への最短経路が複数個あるとき，そのうちの一つだけが木 T の経路として残る.

グラフの指定方法

1. 隣接行列

	s	t	x	y	z
s	–	3	–	5	–
t	–	–	6	2	–
x	–	–	–	–	2
y	–	1	4	–	6
z	3	–	7	–	–

n都市のとき，nxnのメモリが必要．
しかし，２点間に辺があるかどうか
は定数時間で求めることができる．



隣接リスト形式

s → t(3), y(5)
t → x(6), y(2)
x→z(2)
y→t(1), x(4), z(6)
z→s(3), x(7)

辺数だけのメモリですむが，
２点間に辺があるかどうかは
探索しなければならない．

各頂点 v で管理する情報

d[v] = 現在までに得られた始点 s からの最短経路の長さ
parent[v] = 現在までに得られた始点 s からの最短経路での親
status[v] = {未調査, 調査中, 距離確定}
　status[v] = 未調査　　まだ一度も調べていない
　status[v] = 調査中　　現在調査中で, 最短距離は未確定
　status[v] = 距離確定　　vまでの距離が確定した.
頂点の状態は「未到達」から始まり,「調査中」を経て最後に「距離確定」になる. すべての頂点が「距離確定」の状態になれば終了.

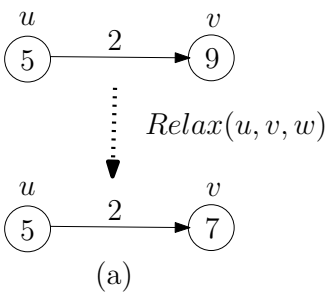
初期設定

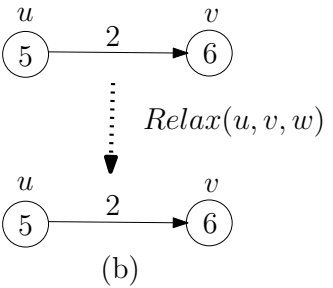
d[v] =　無限大
parent[v] = NIL
status[v] = 未調査
ただし, d[s] = 0. parent[s] = NIL, status[s] = 調査中

基本的な考え方

常に,「調査中」の頂点vの集合をd[v]の値をキーとして
管理しておき, d[v]の値が最小の頂点uを取り出す.
頂点uの状態を「距離確定」に変更する.
頂点uから出る辺(u,v)それぞれについて,
　　この辺を使った方がd[v]の値が改善できるか調べる.
　　改善できるなら, d[v]の値を変更する.
　　if d[u] + w(u, v) < d[v] then d[v] = d[u] + w(u, v)
上記の操作をすべての頂点が「距離確定」の状態になるまで
繰り返す.

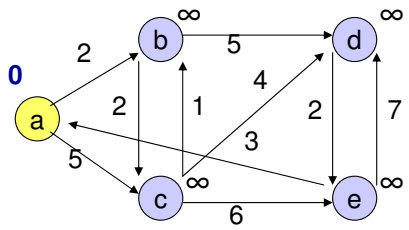
緩和法





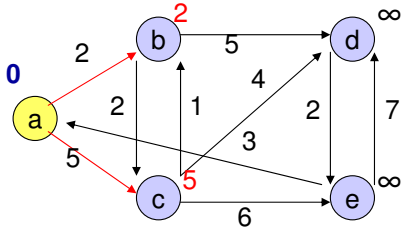
w(u, v)=2の重みをもつ辺に対する緩和操作
始点sから頂点uまでの距離の推定値をd[u]として管理.
(a)の場合
　d[u]=5, d[v]=9, w(u,v)=2のとき，d[v]=5+2=7に改善
(b)の場合
　d[u]=5, d[v]=6, w(u,v)=2のとき，d[v]の値に改善なし

実行例



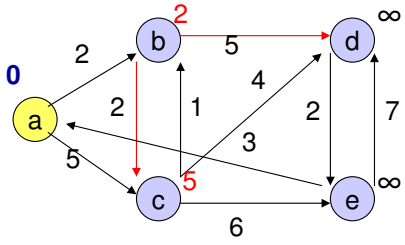
始点　s = a
調査中の頂点={s=a}
距離確定の頂点=空
未調査の頂点={b, c, d, e}

調査中の頂点から a を選択し,
aから出る辺(a, b), (a, c)について
緩和操作を実行



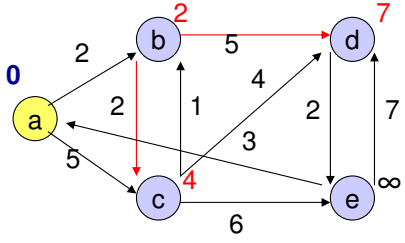
d[b] = 2
d[c] = 5
調査中の頂点={b, c}
距離確定の頂点={a}
未調査の頂点={d, e}

実行例



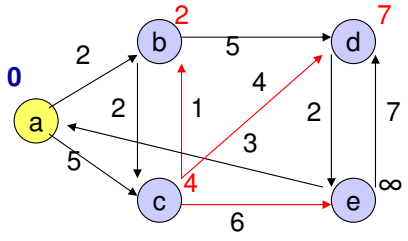
調査中の頂点={b, c}
d[b] = 2, d[c] = 5

調査中の頂点から b を選択し,
bから出る辺(b, c), (b, d)について
緩和操作を実行



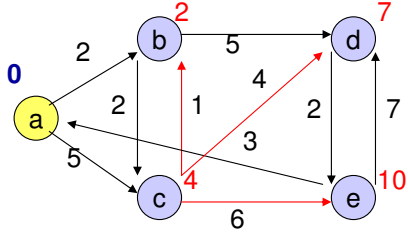
d[c] > d[b]+w(b,c)=2+2 ➔ d[c]=4
d[d] = 2+5 = 7
調査中の頂点={c, d}
距離確定の頂点={a, のb}
未調査の頂点={e}

実行例



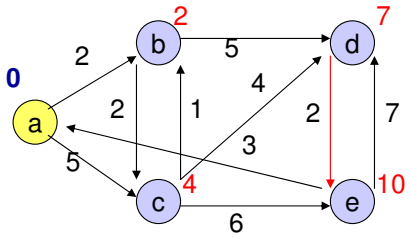
調査中の頂点={c, d}
d[c] = 4, d[d] = 7

調査中の頂点から c を選択し,
cから出る辺(c, b), (c, d), (c, e)に
ついて緩和操作を実行



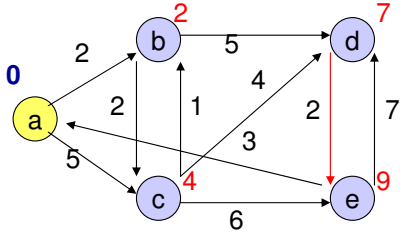
d[b] =2 < d[c]+w(c, b)=4+1
d[d] =7 < d[c]+w(c,d)=4+4
d[e] = 10
調査中の頂点={d, e}
距離確定の頂点={a,b,c}
未調査の頂点={ }

実行例



調査中の頂点={d, e}
d[d] = 7, d[e] = 10

調査中の頂点から d を選択し,
dから出る辺(d, e)について
緩和操作を実行

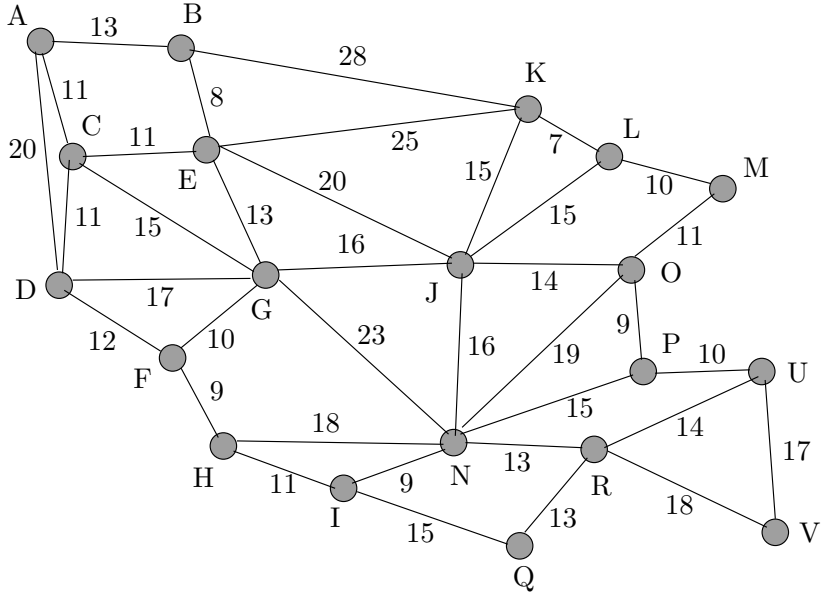


d[e] =10 >d[d]+w(d,e)=7+2=9
➔ d[e] = 9
調査中の頂点={e}
距離確定の頂点={a,b,c,d}
未調査の頂点={ }

実際には，1頂点を除いてすべて距離が確定したので，
この時点で終了することができる.
（最後の緩和操作は必要なし）

演習問題

下記のグラフにs=Aとして先に述べたアルゴリズムを適用せよ.



ダイクストラの最短経路アルゴリズム

初期化操作を行った後，始点sからの距離d(s,v)が求まった頂点の集合をSとして管理しながら，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から出る辺に対して緩和操作を施すことを繰り返す．

```
すべての頂点vについて, d[v]=∞, parent(v)=NILとする.
始点sについて, d[s]=0 とする.
集合Sを空集合に初期化する.
while( V-S が空でない){
   V-Sの中でd[u]の値が最小の頂点uを選ぶ.
   uをSに加える.
   for uから出るすべての辺(u, v) do
       緩和操作relax(u,v)を実行.
}
```

ダイクストラ法の計算時間

単純なデータ構造でV-Sを管理する場合

初期化：O(|V|)時間（頂点数に比例）

すべての頂点が一度だけ選ばれて，その頂点から出る辺について緩和操作を繰り返すから，

繰り返しの回数はO(|V|)回

また，緩和操作は各辺について一度だけ実行され，緩和操作におけるd[.]の値の変更も定数時間でできるから，

全体ではO(|E|)時間（辺数に比例する時間）

後は，V-Sの中からd[.]の値が最小のものを選ぶための時間．

単純にV-Sの中でd[.]の値が最小のものを選ぶなら，1回につきO(|V-S|)時間かかる．したがって，全体では，

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

ダイクストラ法の計算時間(2)

平衡分探索木でV-Sを管理する場合

初期化の時間と繰り返しの回数は同じ.
緩和操作におけるd[.]の値の変更はO(log |V|)時間でできるから,
全体ではO(|E|log |V|)時間
V-Sの中からd[.]の最小値を選ぶための時間もO(log |V|).
したがって, 全体では,
O(|E| log |V|)時間
ということになる.

A.単純なデータ構造でV-Sを管理する場合　O(|E|+|V|^2)
B.平衡分探索木でV-Sを管理する場合　O(|E|log |V|)

|E|=O(|V|^2)なら, 単純なデータ構造が有利
|E|=o(|V|^2)なら, 平衡2分探索木が有利

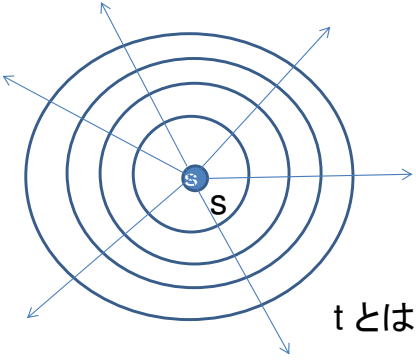
ダイクストラ法の高速化（理論的）

フィボナッチヒープという特殊なデータ構造を用いると，
緩和操作におけるd[.]値の変更が全体でO(|E|)時間で可能
（1回当たりに直すとO(1)時間）．
また，d[.]値の最小値もO(log |V|)時間で求めることが可能．
よって，全体の計算時間は

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

ダイクストラ法における探索

基本的には始点を中心とする円を拡大しながら探索を行う.



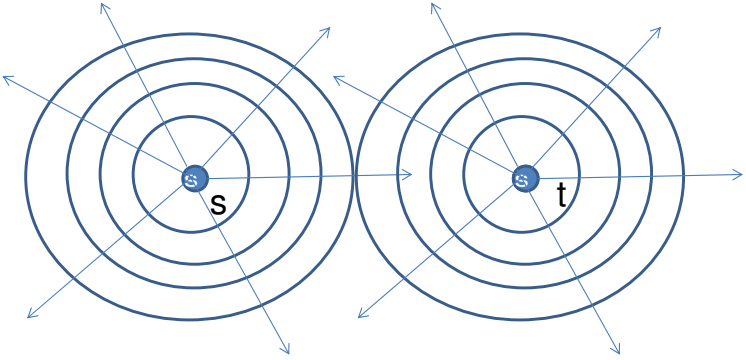
t

t とは逆方向にも探索していることに注意

アルゴリズムで探索のための辺を延ばすとき，目的地 t が考慮されていないので，sから同心円状に探索が進むことに注意.

ダイクストラ法の効率化

始点からだけではなく，目標点からも逆方向に経路を探索．
両方からの探索の円が衝突すれば最短経路が見つかる．



漸近的な計算時間は変わらない（オーダーを改善するほどのものではない）．