講義３：最短経路問題３

上原隆平
uehara@jaist.ac.jp

Bellman-Fordのアルゴリズム

辺の重みとして負の値を許す一般的な最短経路問題に対するアルゴリズム.

グラフと始点が与えられたとき，始点から到達可能な負の重みの閉路が存在すれば，解が存在しないことを報告し，そのような閉路が存在しなければ，最短経路木を返したい.

このアルゴリズムでは，緩和法を用いて，始点sから各頂点vへの最短路の重みに関する推定値　d[v]を実際の最短路重みdist(s, v)になるまで徐々に減らしていく.

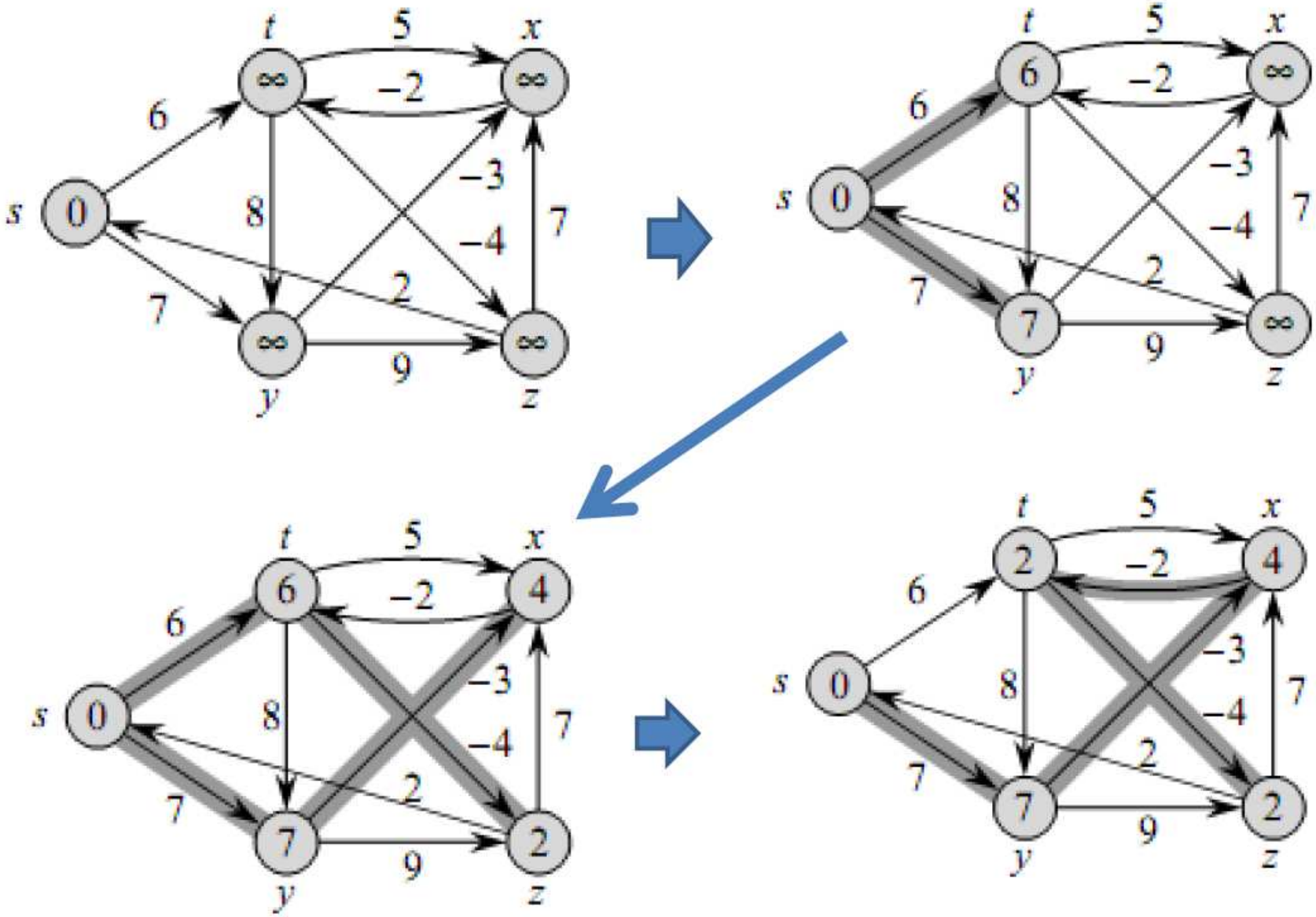
Bellman-Fordのアルゴリズム

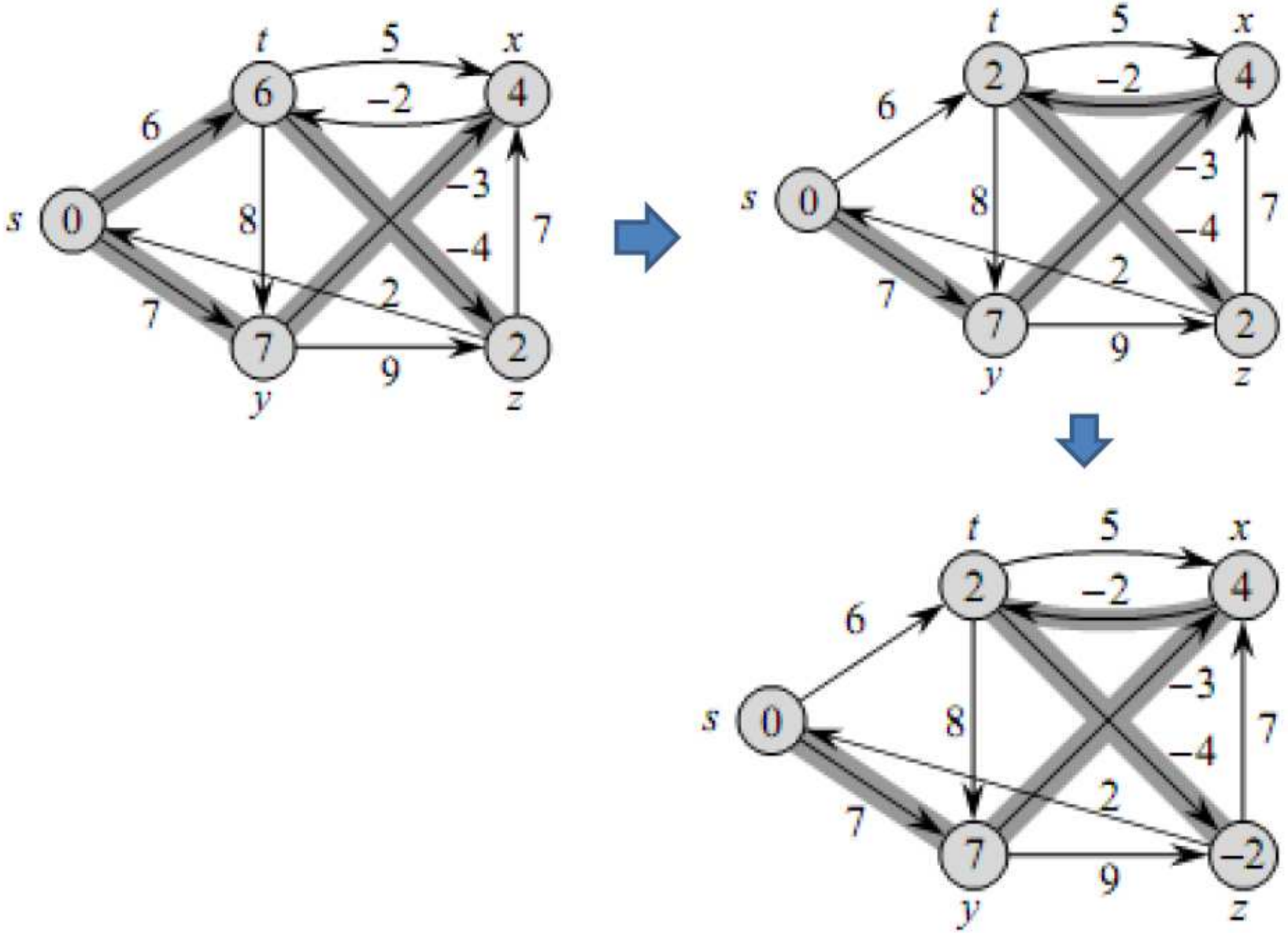
G: 入力のグラフ, s: 始点
始点からの距離の推定値d[v]を∞に設定

```
for i=1 to |V|-1 do
    do for 各辺(u, v)について
        do Relax(u, v, w)

for 各辺(u, v)について
    do if d[v] > d[u] + w(u, v)
        then return FALSE
return TRUE.
```

計算時間はO(VE).

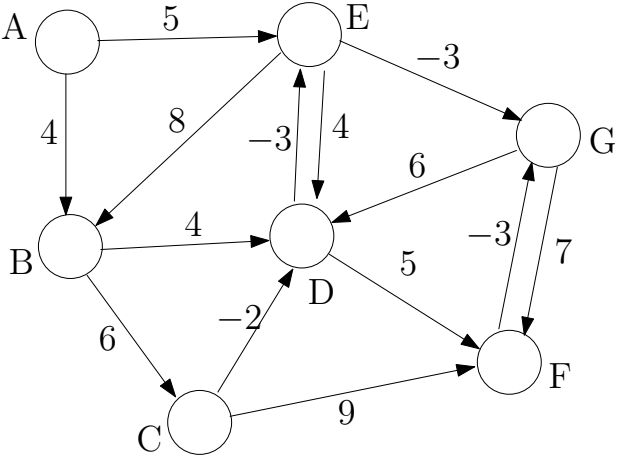

t
x
s
y
z
∞
6
5
−2
−3
8
−4
7
2
9
0
7



問題：調べる辺の順序を変えると効率は改善するか.

演習問題

下記のグラフに，Aを始点としてBellman-Fordのアルゴリズムを適用せよ.

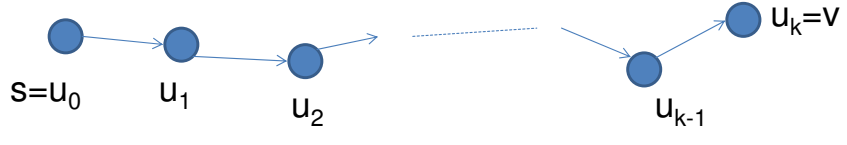


補題：始点から到達可能な負の重みの閉路が存在しないとき，Bellman-Fordのアルゴリズムを適用すると，始点sから到達可能なすべての頂点vについて，

$d[v] = dist(s, v)$

が成り立つ．ただし，dist(s, v)はsからvに至る最短経路の長さ．

証明： sから到達可能な任意の頂点vを考えよう．sからvに至る最短経路を$(s=u_0, u_1, \dots, u_k=v)$としよう．アルゴリズムの最初のループは|V|-1回だけ繰り返されるが，その第 i 回目において辺(u_i, u_{i+1})に対して緩和操作Relaxが実行される．したがって，k回目の繰り返しの後ではd[v]の値はsからvまでの最短経路の長さdist(s, v)に等しくなっている．



補題:始点から到達可能な負の重みの閉路が存在するとき,
Bellman-Fordのアルゴリズムを適用するとFALSEの値が
返される.

証明: 背理法で証明するために, 始点から到達可能な負の
重みの閉路$(u_0, u_1, \dots, u_k)$が存在するが, アルゴリズムでは
TRUEが返されるものと仮定しよう.
アルゴリズムはTRUEを返すから, すべてのu_iについて
$d[u_i] \leqq d[u_{i-1}] + w(u_i, u_{i-1})$, $i=1, 2, \dots, k$
が成り立つ.
上式をi=1, 2, … , kについて加えると,
$\Sigma_{i=1..k}\, d[u_i] \leqq \Sigma_{i=1..k}\, d[u_{i-1}] + w(u_i, u_{i-1})$
$= \Sigma_{i=1..k}\, d[u_{i-1}] + \Sigma_{i=1..k}\, w(u_i, u_{i-1})$
ここで, 閉路$(u_0, u_1, \dots, u_k)$は負の重みを持っていたから,
$\Sigma_{i=1..k}\, w(u_i, u_{i-1})<0$ が成り立つ. よって,
$\Sigma_{i=1..k}\, d[u_i] \leqq \Sigma_{i=1..k}\, d[u_{i-1}] + \Sigma_{i=1..k}\, w(u_i, u_{i-1}) < \Sigma_{i=1..k}\, d[u_{i-1}]$.
$u_0=u_k$だから, $\Sigma_{i=1..k}\, d[u_i] = \Sigma_{i=1..k}\, d[u_{i-1}]$. よって矛盾.

目標に向かう探索を優先： goal-directed search
A* アルゴリズム

ダイクストラのアルゴリズム

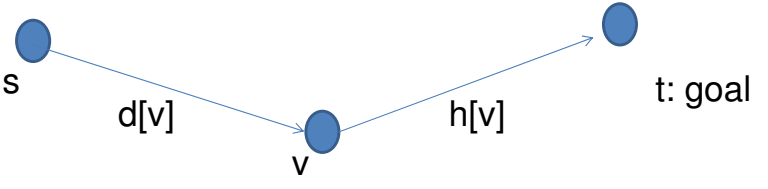
すべての頂点vについて, d[v]=∞, parent(v)=NILとする.
始点sについて, d[s]=0 とする.
集合Sを空集合に初期化する.
while(V-S が空でない){
　V-Sの中でd[u]の値が最小の頂点uを選ぶ.
　u = t のとき終了.
　uをSに加える.
　for uから出るすべての辺(u, v) do
　　緩和操作relax(u,v)を実行.
}

次に選択する頂点の基準を変更.

次の頂点を選択する基準

ダイクストラ法： 始点からの距離の推定値 d[v]

A*アルゴリズム： 始点からの距離の推定値d[v]
＋目標点までの距離の推定値 h[v]



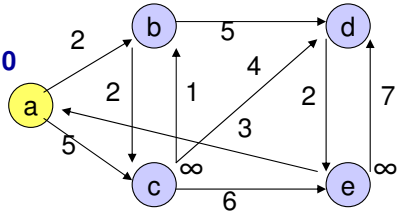
理論的な保証

目標点までの距離の推定値 h[v] が目標点までの
実際の距離より大きくなることがないなら，この方法で
最短経路が必ず見つかる.

A*アルゴリズム

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]+h[v]の値が最小の頂点uを選ぶ.
    u = t のとき終了.
    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.
}

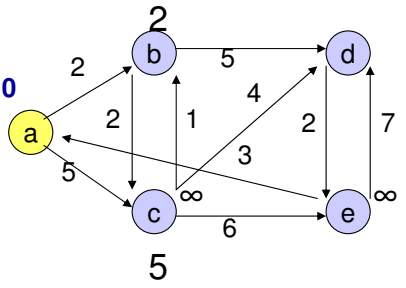
もちろん, すべての頂点vについてh[v]=0とすれば,
ダイクストラ法と同じである.



目標点eまでの距離の下界が分かっているものとする.

h[a] = 5
h[b] = 5
h[c] = 6
h[d] = 2

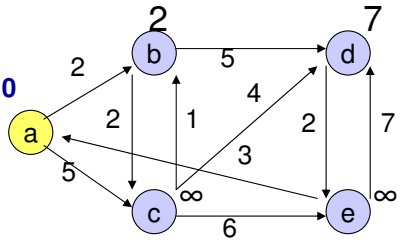
始点 = a
目標点 = e
最初はaを選択し, aから出る辺(a,b), (a,c)について緩和操作:
d[b] = 2, d[c] = 5
調査中={b, c}
d[b]+h[b] = 2+5=7
d[c]+h[c] = 5+6=11
よって, 次はbを選択.



目標点eまでの距離の下界が分かっているものとする.

h[a] = 5
h[b] = 5
h[c] = 6
h[d] = 2

よって, 次はbを選択.
bから出る辺(b,c), (b,d)について緩和操作を行う
d[d] = 2+5=7
d[c] = 5 > d[b] + w(b,c)=2+2=4
➔d[c] = 4
調査中 = {c, d}
d[c]+h[c] = 4+6=10
d[d]+h[d] = 7+2 = 9
よって, dを選択.



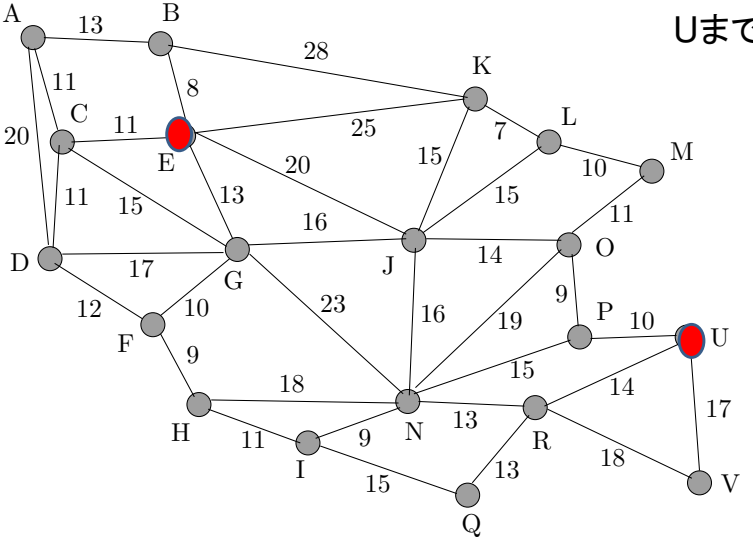
目標点eまでの距離の下界が分かっているものとする.

h[a] = 5
h[b] = 5
h[c] = 6
h[d] = 2

よって, 次はdを選択.
dから出る辺(d, e)について
緩和操作を行う
d[e] = d[d] + w(d, e)=7+2=9

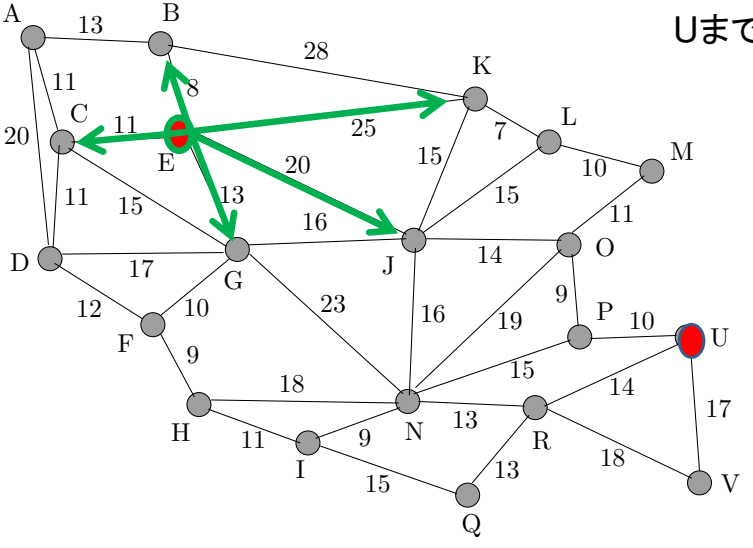
これで目標点までの最短経路の候補が見つかったが, この後, これが実際に最短であることを保障する後処理が必要
(場合によっては, 後処理でもっと短い経路が見つかることもある).

後処理のため, 最短経路の候補が見つかった後も同じ処理を続ける.
目標点が選ばれた時点で最短経路が見つかっている



Uまでの距離の下界

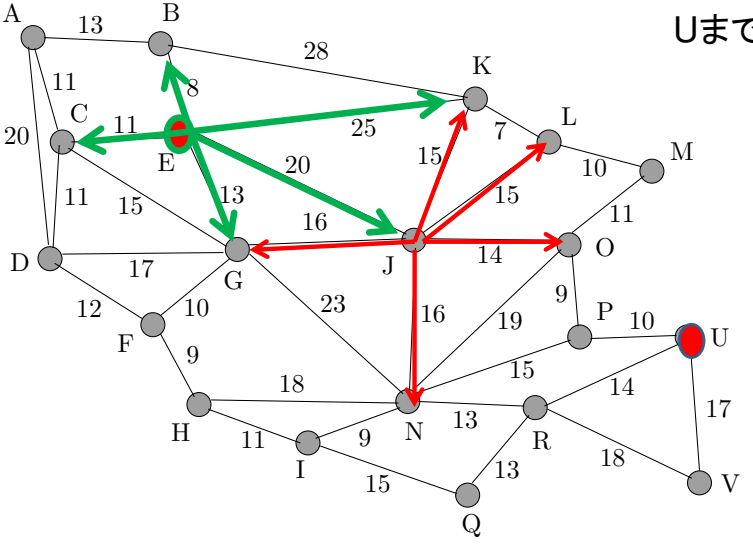
A	45	O	12
B	42	P	10
C	43	Q	26
D	42	R	14
E	33	U	0
F	35	V	17
G	29		
H	33		
I	32		
J	19		
K	23		
L	20		
M	15		
N	23		



Uまでの距離の下界

A	45	O	12
B	42	P	10
C	43	Q	26
D	42	R	14
E	33	U	0
F	35	V	17
G	29		
H	33		
I	32		
J	19		
K	23		
L	20		
M	15		
N	23		

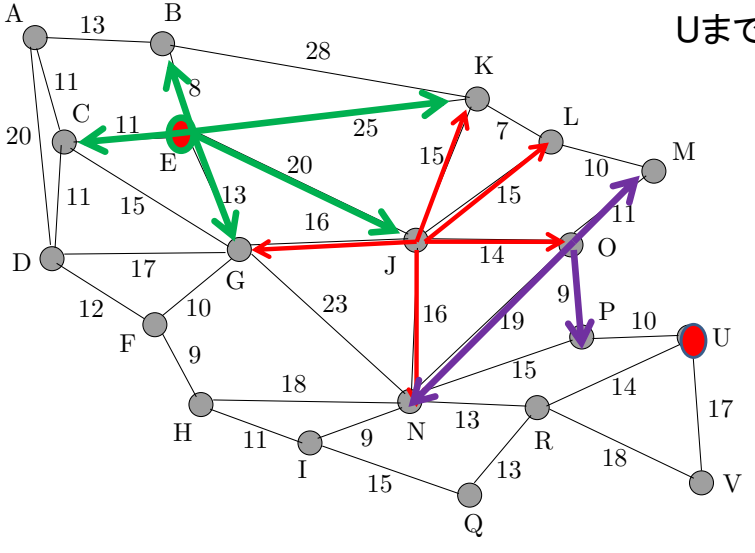
B: 8+42=50, C: 11+43=54, G: 13+29=42
J: 20+19=39, K: 25+23=48
Jを選択.



Uまでの距離の下界

A	45	O	12
B	42	P	10
C	43	Q	26
D	42	R	14
E	33	U	0
F	35	V	17
G	29		
H	33		
I	32		
J	19		
K	23		
L	20		
M	15		
N	23		

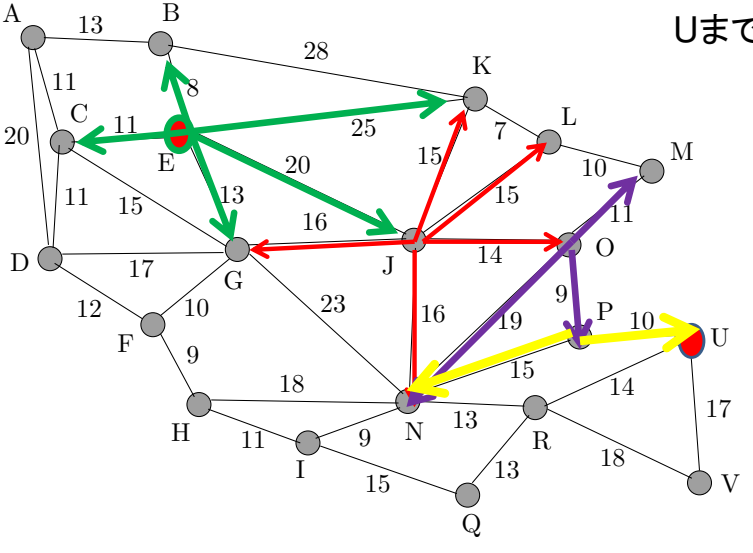
G: 20+16+29=65, K: 20+15+23=58,
L: 20+15+20=55, N: 20+16+23=59,
O: 20+14+12=46　Oを選択



M: 20+14+11+15=61,
N: 20+14+19+23=76,
P: 20+14+9+10=53.

Uまでの距離の下界

A	45	O	12
B	42	P	10
C	43	Q	26
D	42	R	14
E	33	U	0
F	35	V	17
G	29		
H	33		
I	32		
J	19		
K	23		
L	20		
M	15		
N	23		

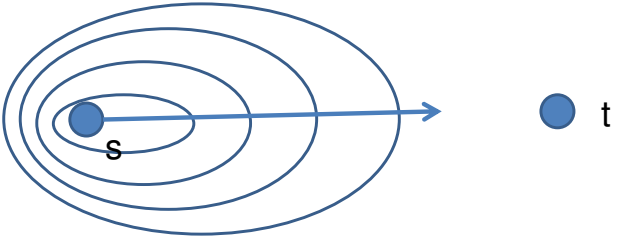


U: 20+14+9+10=53,
N: 20+14+9+15=58
次にUが選ばれるので，これが最短路.

Uまでの距離の下界

A	45	O	12
B	42	P	10
C	43	Q	26
D	42	R	14
E	33	U	0
F	35	V	17
G	29		
H	33		
I	32		
J	19		
K	23		
L	20		
M	15		
N	23		

A*法の概念図

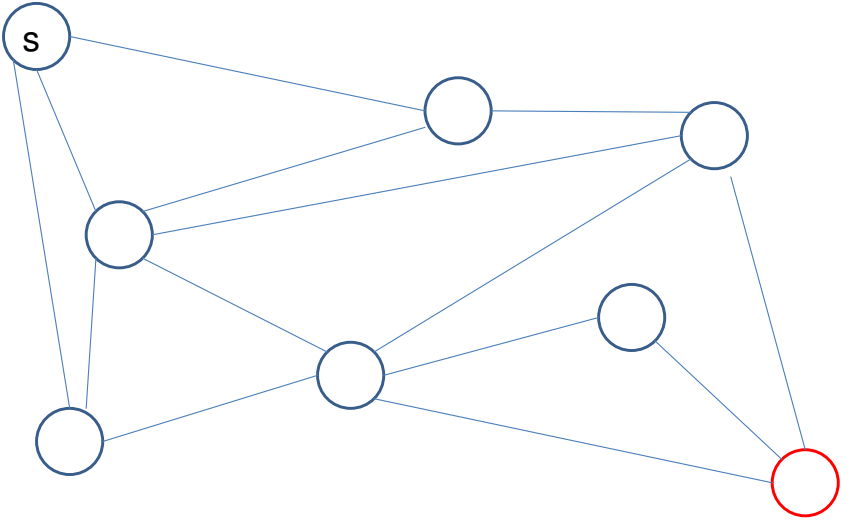


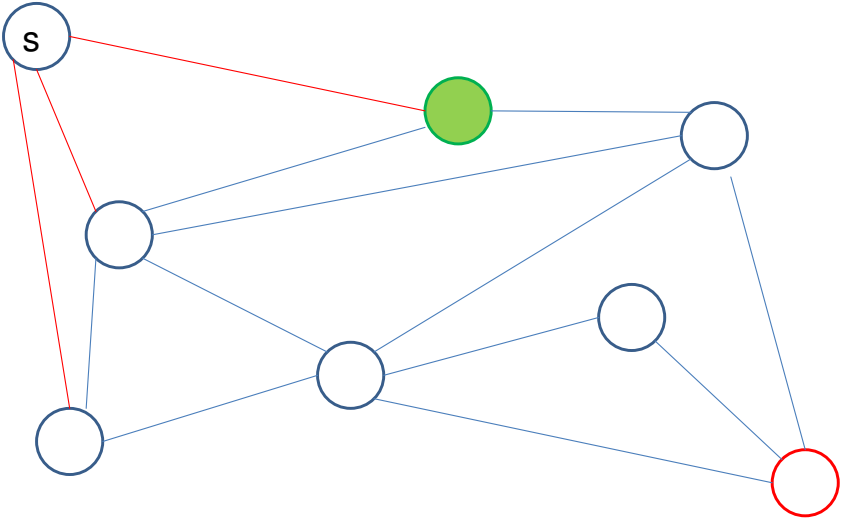
目標点までの距離の下界値？

自動車のナビゲーションシステムでは，始点と目標点を指定した最短経路探索が行われる．
現在地点から目標点までの直線距離は確かに目標点までの距離の下界値になっている．

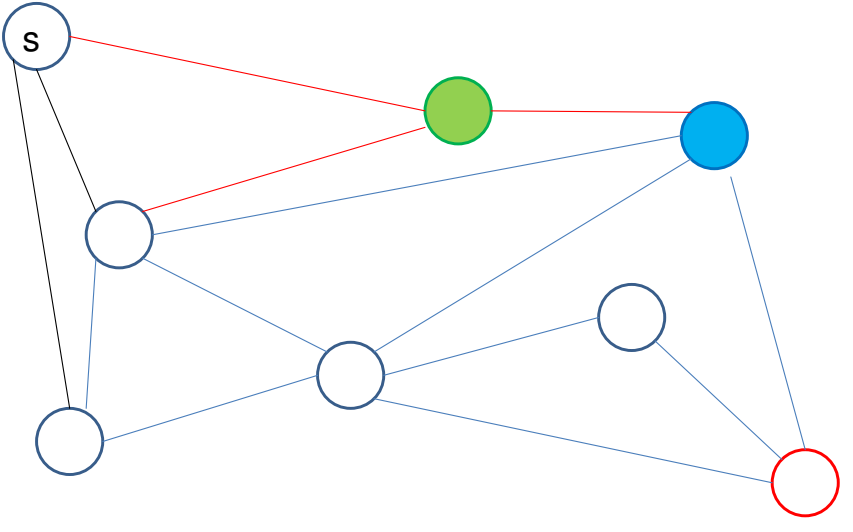
目標点までの距離の下界値h[v]が実際の距離に近いほど探索の効率は良くなる．実際，目標点までの距離が正確に分かっている場合には，A*は一直線に目標点を目指す．

距離ではなく最も時間の短い経路を求めるとき，下界はどのように定めればよいか？

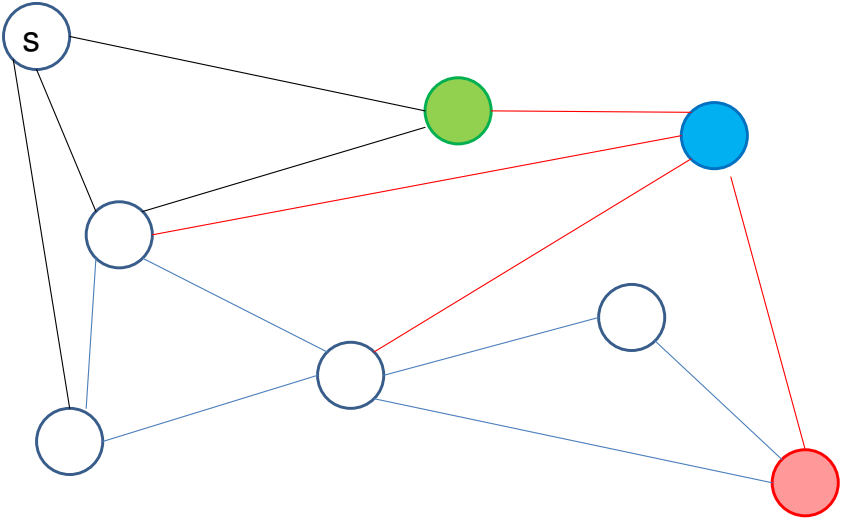

s



どの頂点が目標点に最も近いか？



どの頂点が目標点に最も近いか？



どの頂点が目標点に最も近いか？

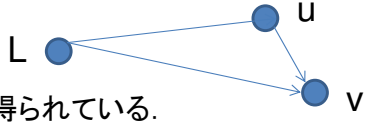
ランドマーク法

この方法ではいくつかの頂点をランドマークとして選んでおき，
前処理として，それらの頂点から各頂点までの距離を求めておく．

Lをランドマークとし，頂点uとvについて，Lからの距離をd(L, u)とd(L, v)で表すものとする．このとき，三角不等式により，
u-v間の距離d(u, v)について

$$d(L, u) - d(L, v) \leqq d(u, v)$$

が成り立つ．
この式はどのランドマークについても成り立つ．よって，すべてのランドマークについて上記の（左辺）値の最大値を取れば，それはu-v間の距離の良い下界を与えている．



どの頂点が終点として与えられても，
任意の頂点からその点までの距離の良い下界が得られている．

ランドマーク法

目的地 t が与えられたとき，各頂点vについて，

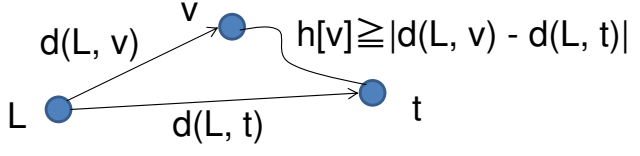
max{ |d(L, v) - d(L, t)| }

を求めると，これがA*アルゴリズムで必要な目的地 t までの距離の下界値になっている．

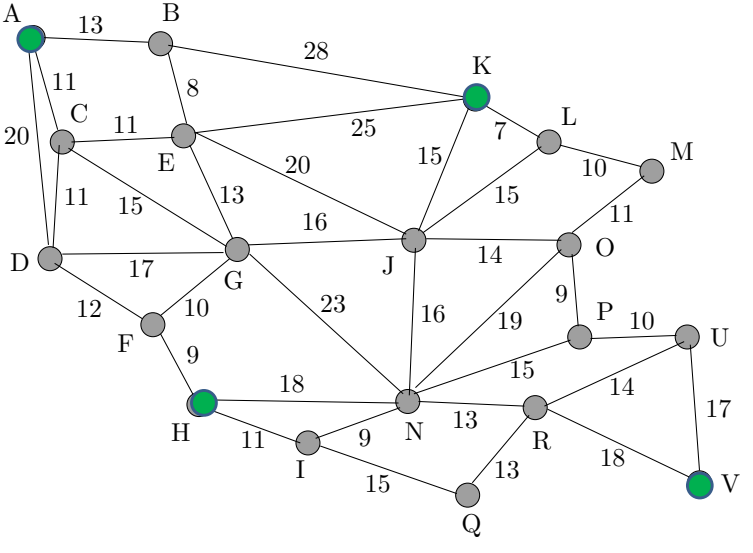
よって，

h[v] = max{ |d(L, v) - d(L, t)| }

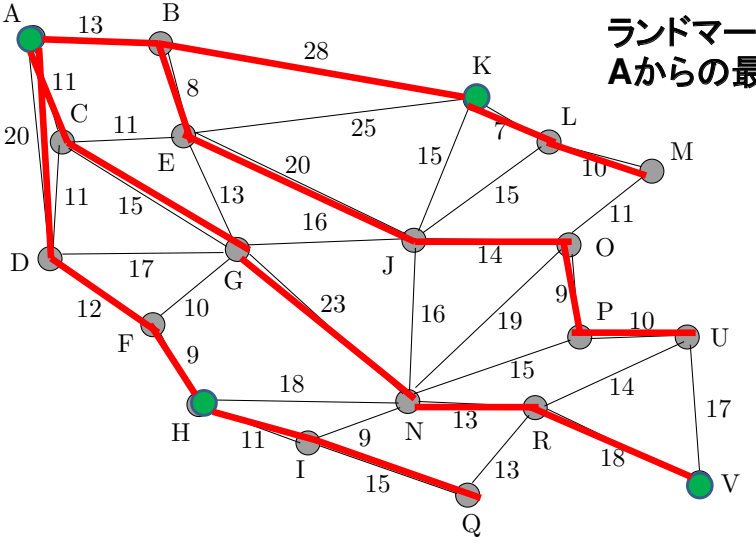
としてA*アルゴリズムを適用すると，効率が良い．



たまたまランドマーク地点が目的地点として選ばれれば
最短経路が見つかったことになる．

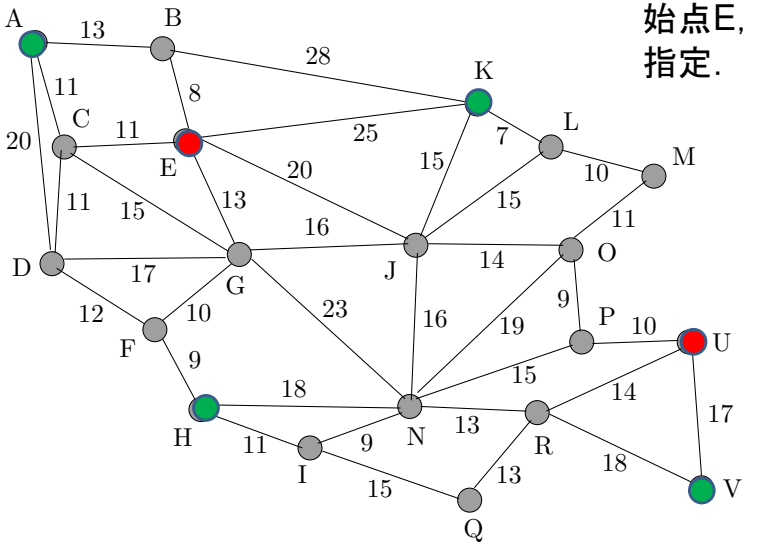


A, H, K, Vの4頂点をランドマークとして選んでおき，それらの頂点からの最短距離を求めておく.



ランドマーク
Aからの最短経路木

A, H, K, Vの4頂点をランドマークとして選んでおき，それらの頂点からの最短距離を求めておく.



始点E，目的地Uを指定.

Eの近傍頂点vで d[v] + h[v]の値を計算.

h[v] = max(d(Landmark, v) - d(Landmark, U)): Jについては，|d(A, J)-d(A, U)|=|41-74|=33, |d(H,J)-d(H,U)|=|34-45|=11, |d(K,J)-d(K,U)|=|15-47|=32, |d(V,J)-d(V,U)|=|47-17|=30.