

bit全探索 bitDP

bit全探索とは

- bit演算を用いて 2^n 通りの状態を全探索するテクニック
 - 「オン/オフ」, 「使う/使わない」, 「買う/買わない」などの 2 通りの選択肢が複数ある状況で有効
 - 2 通りの選択肢を 0/1 で表現する
- あり得る全ての状況を列挙して, それぞれの場合において計算や判定を行う

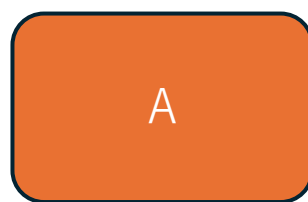
商品 3	商品 2	商品 1
買う	買わない	買う



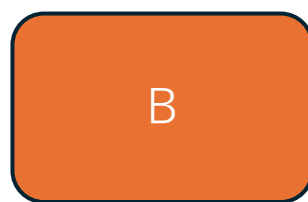
商品 3	商品 2	商品 1
1	0	1

具体例

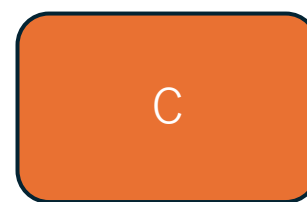
- ・ 3つの商品A,B,Cの中からいくつか選んで金額を100円にできるか？



90円



50円



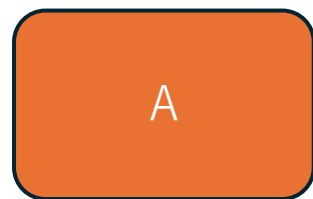
10円

方針: 選び方を全て試して金額が100円になる選び方があるか探したい

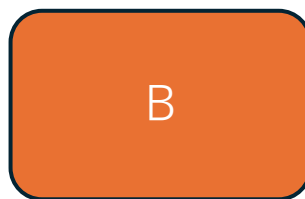
=>A,B,Cそれぞれ選ぶ選ばないの2通りあるので $2^3 = 8$ 通り試せば良い

選び方を列挙する

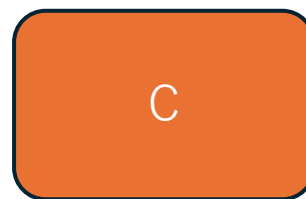
- ・ 3つの商品A,B,Cの中からいくつか選んで金額を100円にできるか？



90円



50円



10円

選んだを1,選ばなかったを0とすると...

	A	B	C	合計金額
0	0	0	0	0円
1	0	0	1	10円
2	0	1	0	50円
3	0	1	1	60円
4	1	0	0	90円
5	1	0	1	100円
6	1	1	0	140円
7	1	1	1	150円

←できた

bit演算

- bit全探索ではbit列を扱う
→ どうやって実装する？

- 計算機上で整数は全て 2 進数で計算されている

0 → 0...000 1 → 0...001 2 → 0...010 3 → 0...011
4 → 0...100 5 → 0...101 6 → 0...110 7 → 0...111

- さらに 2^n 通りの状態は $0, 1, \dots, 2^n$ (10進数) で表せる

bit演算

bit全探索で使用するbit演算

ビットシフト

```
n = 14  # 1110(2)
print(n << 1)
#> 28  (11100(2))
print(n >> 2)
#> 3   (11(2))
```

```
n      = 1110|
n<<1 = 11100|
n>>2 =   11|10
```

bit論理積

```
a = 14  # 1110(2)
b = 11  # 1011(2)
print(a & b)
#> 10  (1010(2))
```

```
1110
&1011
----
1010
```

bit演算

右から

- i bit目の求め方
- $n \gg i$
 - n を i 個右シフト
 - n の i -bit目が 1 bit目に来る
- $n \& 1$
 - n と $0\cdots001$ のbit論理積
 - 1-bit 目が 1 の時に 1, 0 のときに 0 になる

```
n      = 1110|
n>>1  =  111|0
```

```
  111
& 001
----
  001
```

i -bit目の取得

```
bit = 14  # 1110(2)
for i in range(4):
    print(bit >> i & 1)
#> 0
#> 1
#> 1
#> 1
```

例題：問題

問題文

N 個の整数 A_0, A_1, \dots, A_{N-1} と整数 W が与えられます.

A_0, A_1, \dots, A_{N-1} の中からいくつか選んで総和を W にすることはできますか？

制約

- $1 \leq N \leq 20$
- $0 \leq A_i \leq 10^9$
- $0 \leq W \leq 10^9$

問題文

N 個の整数 A_0, A_1, \dots, A_{N-1} と整数 W が与えられます.

A_0, A_1, \dots, A_{N-1} の中からいくつか選んで総和を W にすることはできますか？

制約

- $1 \leq N \leq 20$
- $0 \leq A_i \leq 10^9$
- $0 \leq W \leq 10^9$

例題：解法

- A_0, A_1, \dots, A_{N-1} のそれぞれは「使う/使わない」の 2 通り
- bit全探索でそれぞれを「使う/使わない」を列挙する
- 各状態で使うものの総和を計算して, W と一致するか判定する

例題：実装 (Python)

- 2^n 通りの状態は for 文を使うと楽に表せる
- 各状態において i -bit目が 0 か 1 かは $\text{bit} \gg i \ \& \ 1$ で求められる

計算量 $O(n2^n)$

```
ans = False
for bit in range(1 << n):
    s = 0 # a の部分和

    for i in range(n):
        # i-bit目が 1 かどうか
        if bit >> i & 1:
            s += a[i]

    # 部分和が w と一致するかどうか
    if s == w:
        ans = True
```

bit全探索おまけ

- i bit目が0の時True(1)になってほしい時 n を反転させると楽
 $\sim n \gg i \& 1$

当然 $n \gg i \& 1 == 0$ とかでもいい

- bitシフト演算子の演算順位が低いので注意する
例) $2^n - 1$ を計算したい

```
n = 3;  
print(1 << n - 1); //4  
print((1 << n) - 1); //7
```

練習問題

ABC045C-たくさんの数式

+を入れる場所をbit全探索します。
文字列→整数の変換も確認しましょう

ABC128C-Switches

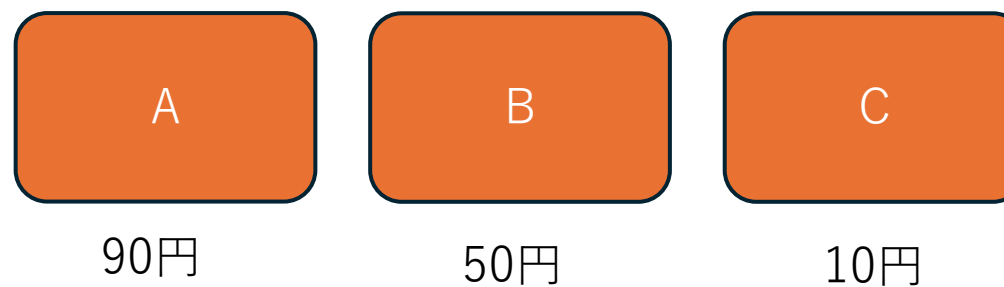
スイッチのON/OFFをbit全探索します。
全ての電球を点灯できるかその都度確認しましょう。

bit DP

bitDPとは

- DPの添字で集合を管理するDP

添字で集合を管理とは？



選んだを1,選ばなかったを0とすると...

「5」で {A,C} という集合
を表現できる

	A	B	C	合計金額
5	1	0	1	100円
6	0	1	1	140円
7	1	1	1	150円

よく bitDP を使う 問題

- ・ 巡回セールスマン問題

グラフが与えられて、全ての頂点を1回ずつ通って戻って来る
最短経路長を求める

-> 例題で扱います

- ・ 和集合を考えていく問題

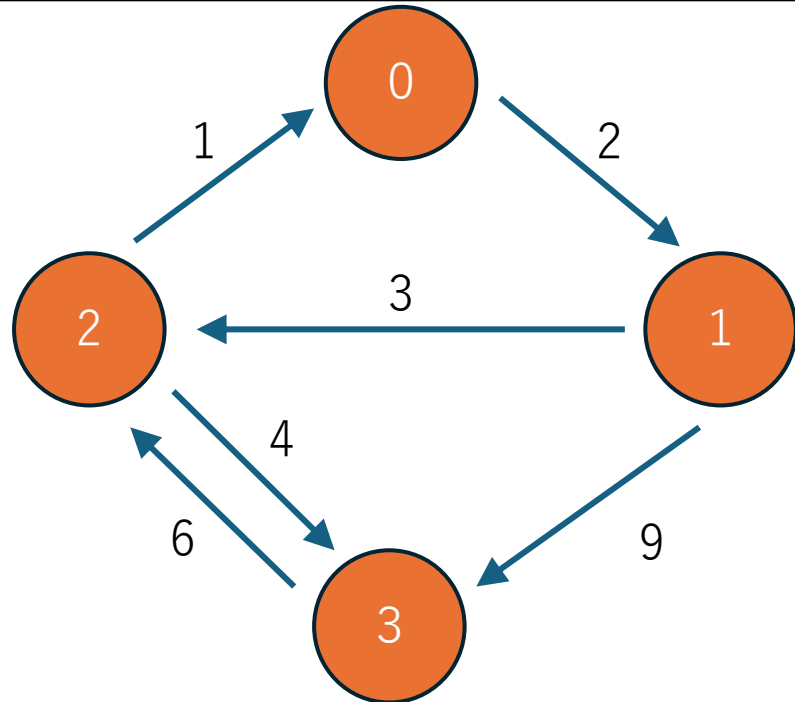
-> 練習問題で

巡回セールスマン問題を考える

巡回セールスマン問題

重み付き有向グラフ $G(V, E)$ について、以下の条件を満たす最短経路の距離を求めて下さい：

- ある頂点から出発し、出発点へ戻る閉路である。
- 各頂点をちょうど1度通る。



方針の前に...

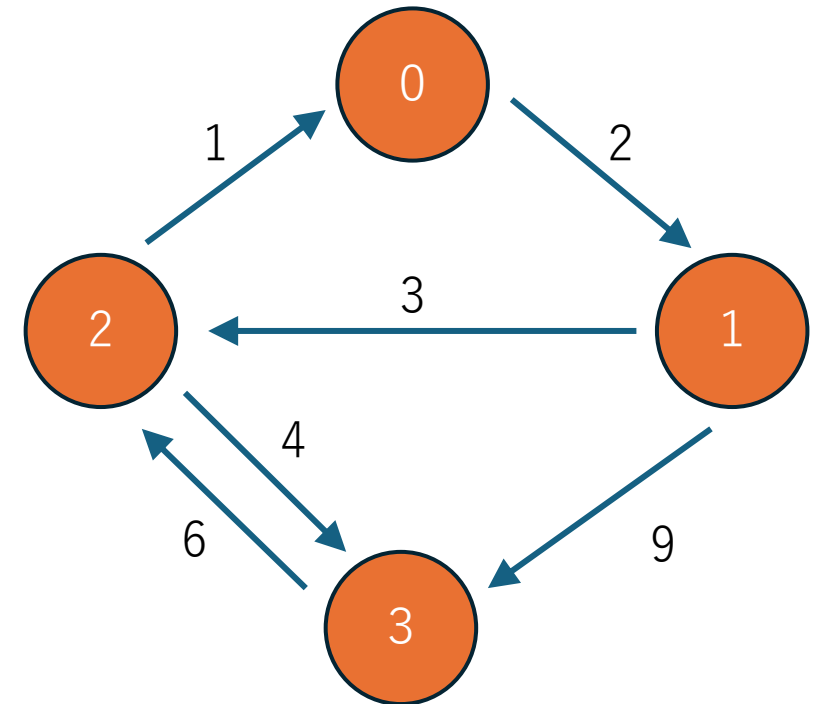
巡回セールスマン問題

重み付き有向グラフ $G(V, E)$ について、以下の条件を満たす最短経路の距離を求めて下さい：

- ある頂点から出発し、出発点へ戻る閉路である。
- 各頂点をちょうど1度通る。

各頂点をちょうど1度通る(一筆書きになる)ので、
どの頂点から考えても良い

->頂点0から考える



方針

巡回セールスマン問題

重み付き有向グラフ $G(V, E)$ について、以下の条件を満たす最短経路の距離を求めて下さい：

- ある頂点から出発し、出発点へ戻る閉路である。
- 各頂点をちょうど1度通る。

頂点0からスタートして

- $dp[i][j] =$ 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含めない)

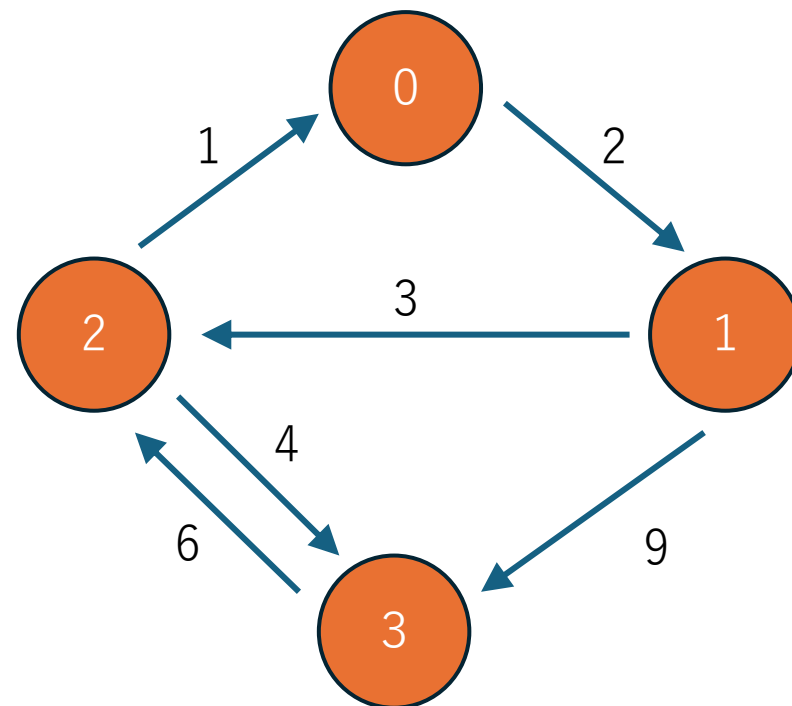
遷移は

ある頂点 $x = \begin{cases} \text{頂点 } j \text{ に隣接} \\ \text{集合 } i \text{ に含まれない頂点} \end{cases}$ に対して

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j, x)|)$$

答えは $dp[(1 \ll n) - 1][0]$ (全頂点通って最後に0についた時)

$$(1 \ll n) - 1 = \underbrace{111\dots11}_{n \text{ 回}} (2)$$

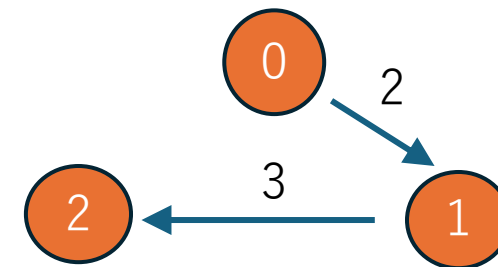


補足1

$dp[i][j]$ = 頂点0からスタートして
今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値

例) $dp[6][2]$ = 頂点1と2を通過して2に着いた時に一番短い経路長

$6=110(2)$

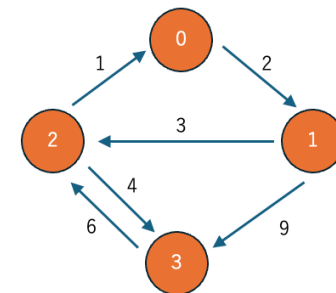


遷移は
ある頂点 x = $\begin{cases} \text{頂点 } j \text{ に隣接} \\ \text{集合 } i \text{ に含まれない頂点} \end{cases}$ に対して

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$

$i + (1 \ll x)$ の例) $i = 5, x = 1$

$$5 + (1 \ll 1) = 0101 + 0010 = 0111(2)$$

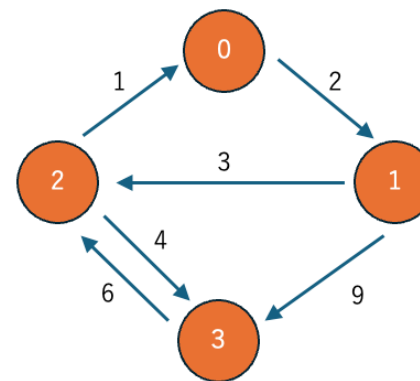


補足2

答えは $\text{dp}[(1 \ll n)-1][0]$ (全頂点通って最後に0についた時)
 $(1 \ll n)-1 = \underbrace{111\dots11}_n (2)$

今回なら $\text{dp}[(1 \ll 4)-1][0]$

$$(1 \ll 4)-1 = 15 = 1111(2)$$



$(1 \ll n)-1$ とすると n 個のビットが立った状態になり全ての頂点を通ったを表現できる！

実装

スライドに載らないので下のリンクから見てください

[Pythonの実装](#)

[C++の実装](#)

g は隣接リスト

```
dp[1<<n][n] = 全ての要素をINFで初期化  
#どこにも訪れてなくて0にいる状態で初期化  
dp[0][0] = 0
```

```
for i = [0..1<<n):  
    for j = [0..n):  
        if dp[i][j]がINFのまま(到達しなかった):continue
```

```
        for (to,cost) in g[j]:  
            if iのtoビット目が1 :continue #すでに通った  
            #遷移  
            dp[i +(1<<to)][to] = min(dp[i + (1<<to)][to],dp[i][j] + cost)
```

答えは

```
if dp[(1<<n)-1][0]がINF => -1  
else dp[(1<<n)-1][0]
```

DPの動き

初期化 (x=INF)

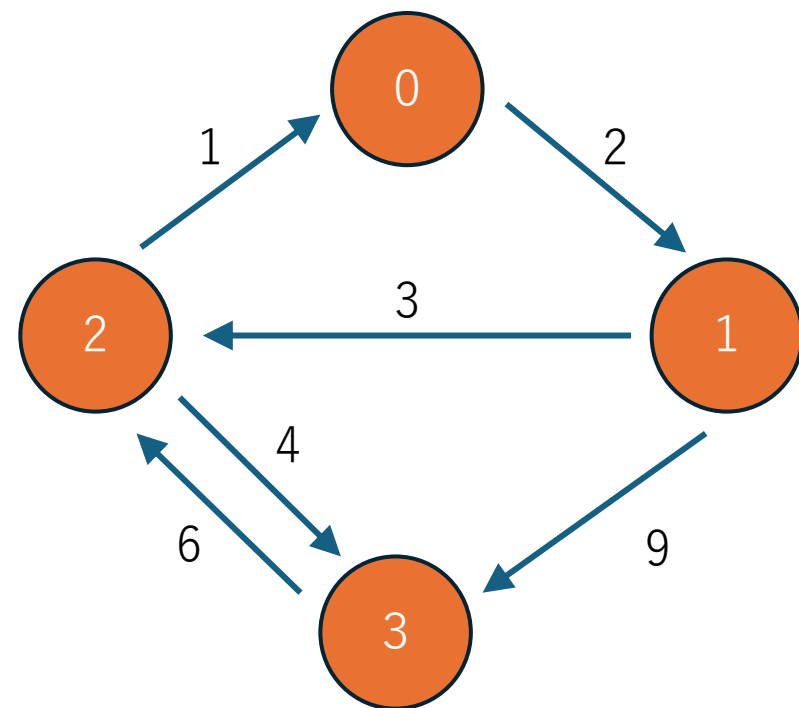
	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	x	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	x	x	x
7(0111)	x	x	x	x
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	x	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	x	x	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$

$dp[1 \ll n][n]$ = 全ての要素を **INF** で初期化
 $dp[0][0] = 0$



DPの動き

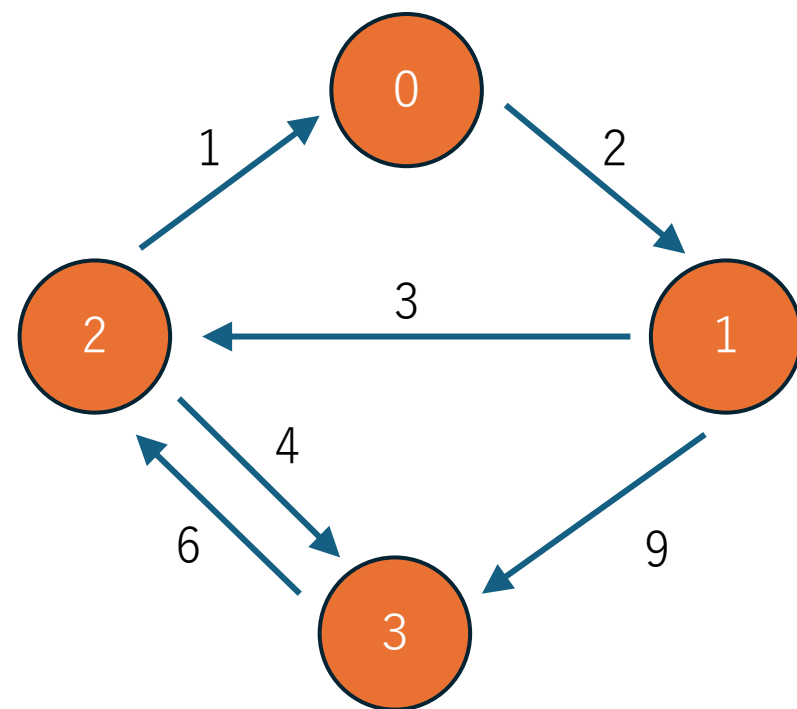
i = 0

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	x	x	x
7(0111)	x	x	x	x
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	x	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	x	x	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

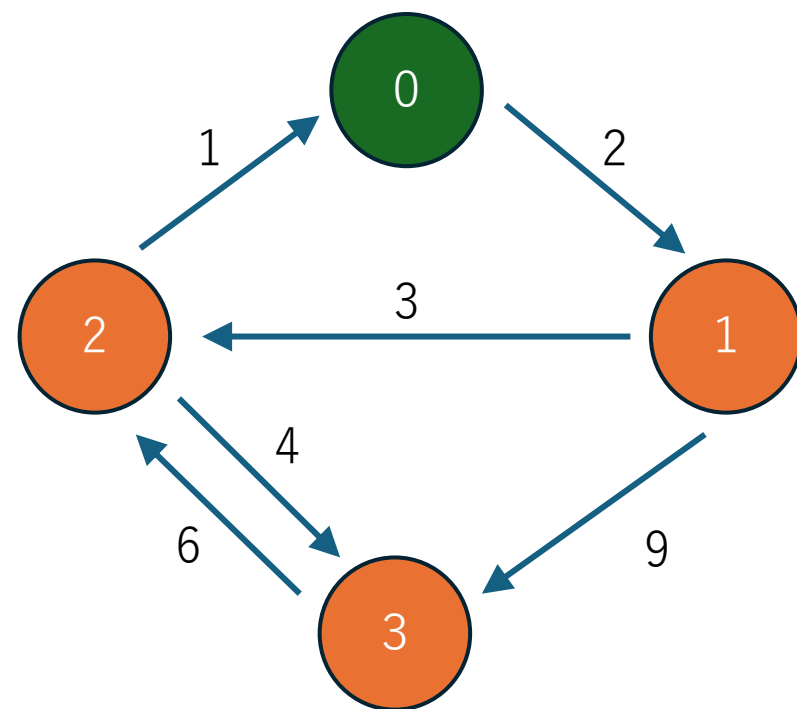
i = 1

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	x	x	x
7(0111)	x	x	x	x
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	x	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	x	x	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

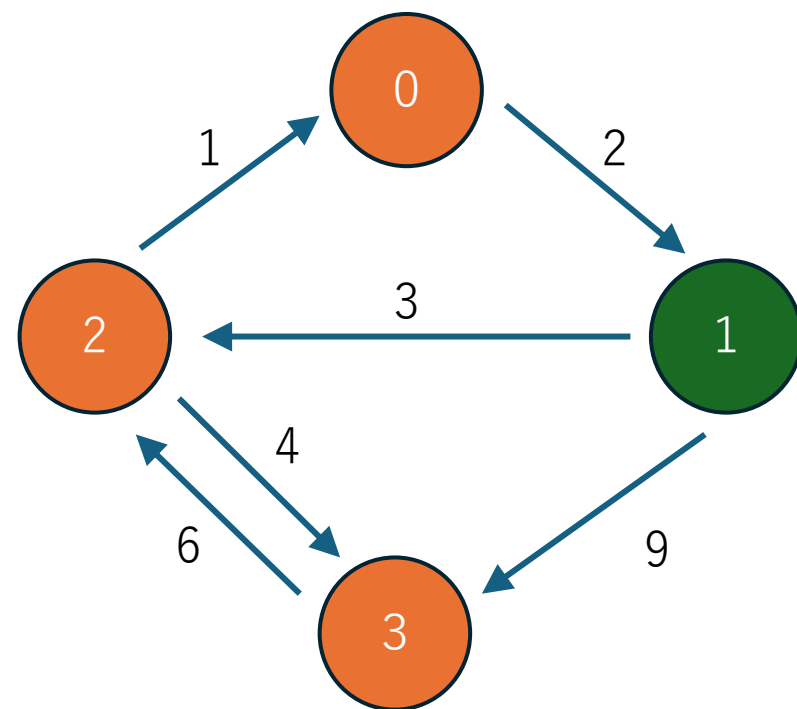
i = 2

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	x
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	x	x	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

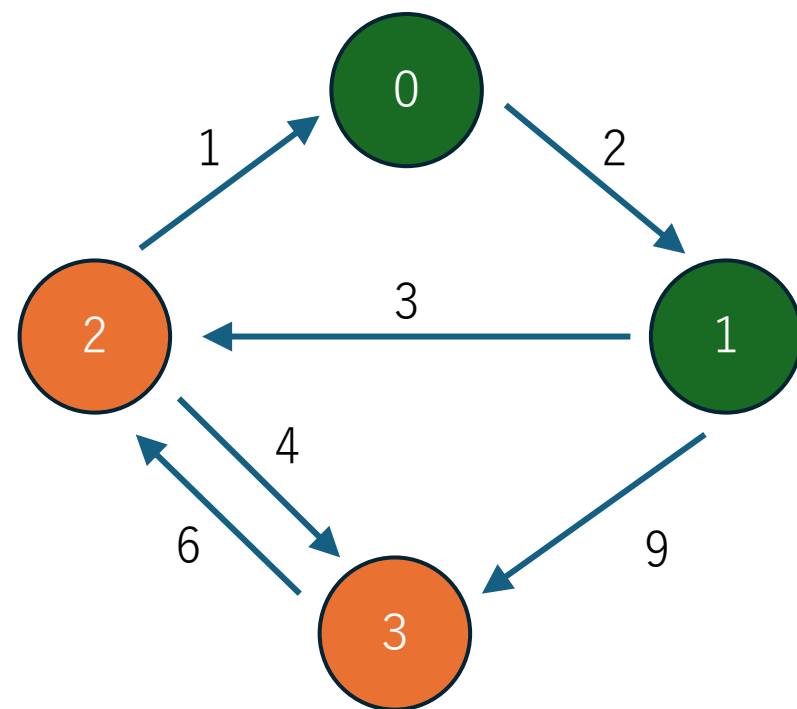
i = 3

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	x
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	x	x	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

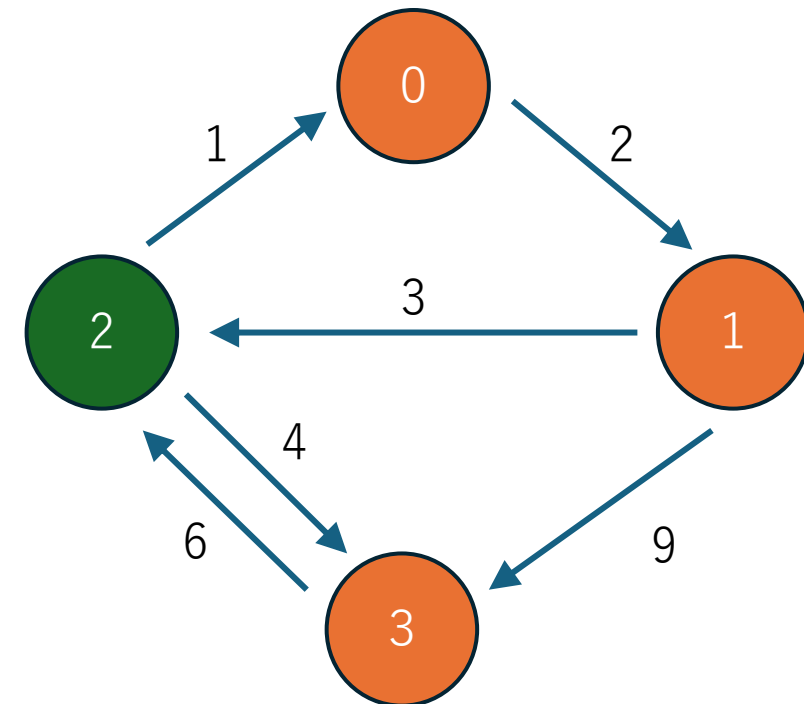
i = 4

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	x
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	x	x	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

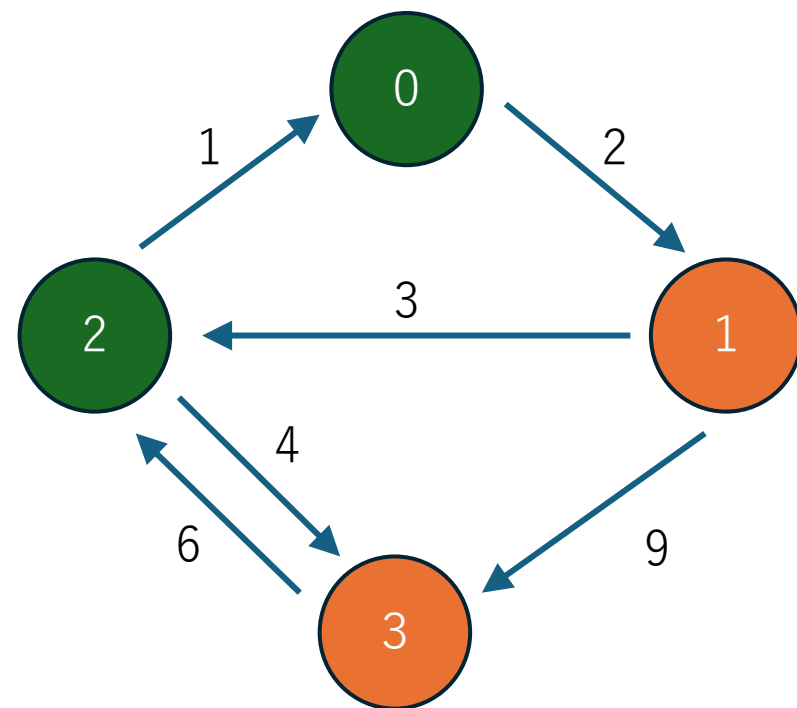
i = 5

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	x
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	x	x	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

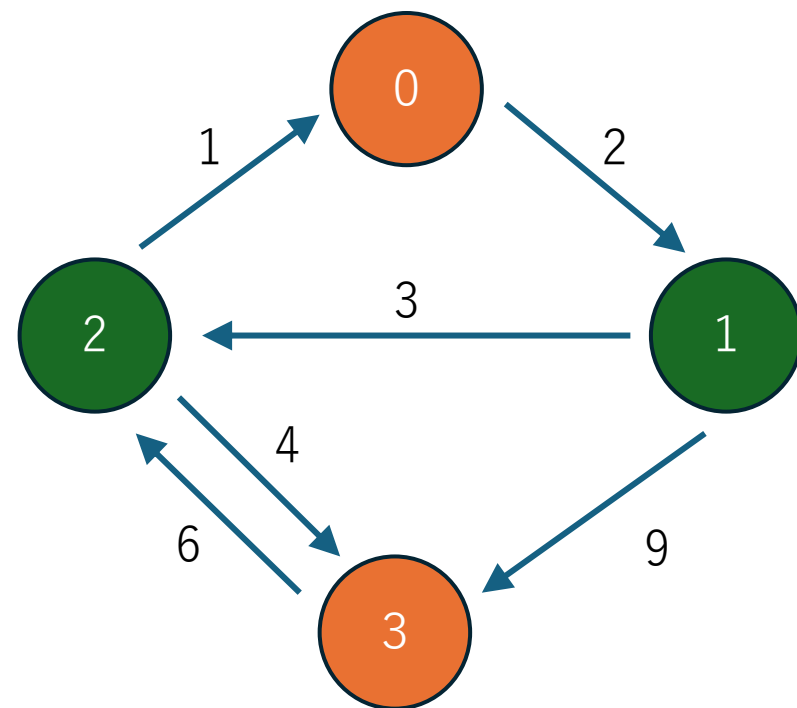
i = 6

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	6
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	9	x	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

i = 7

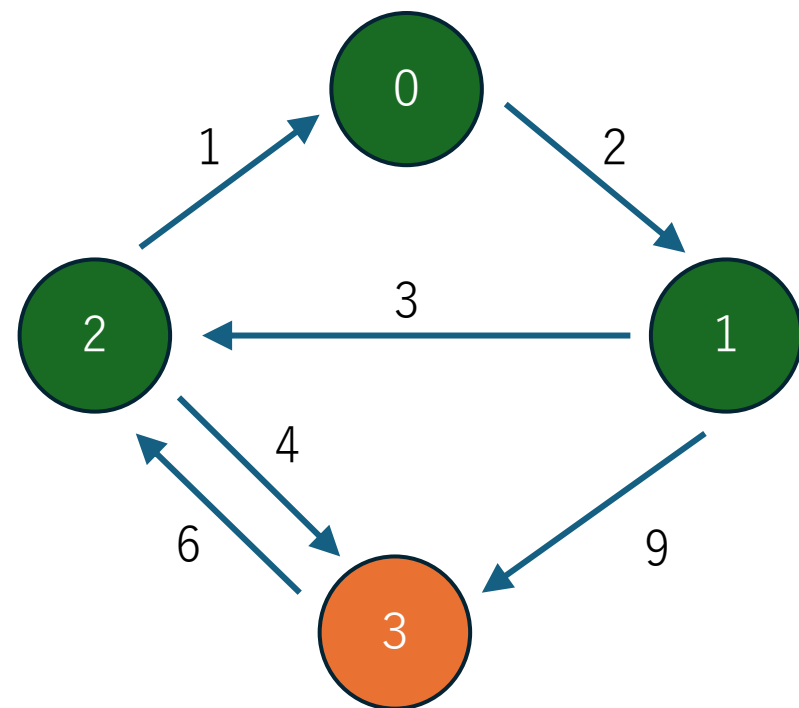
	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	6
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	9	x	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$

```
for (to, cost) in g[j]:
    if iのtoビット目が1 : continue #すでに通った
    #遷移
```



DPの動き

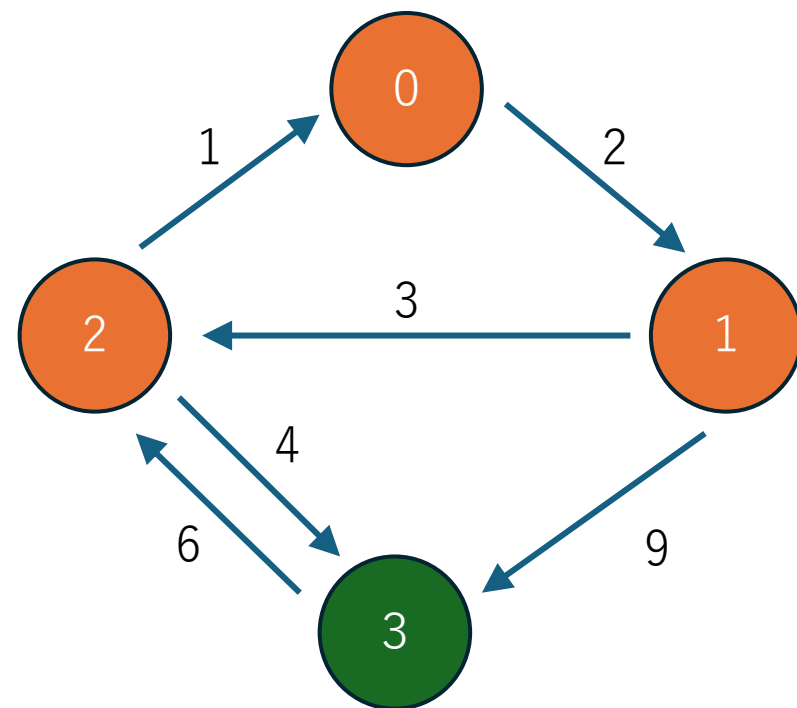
i = 8

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	6
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	9	x	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

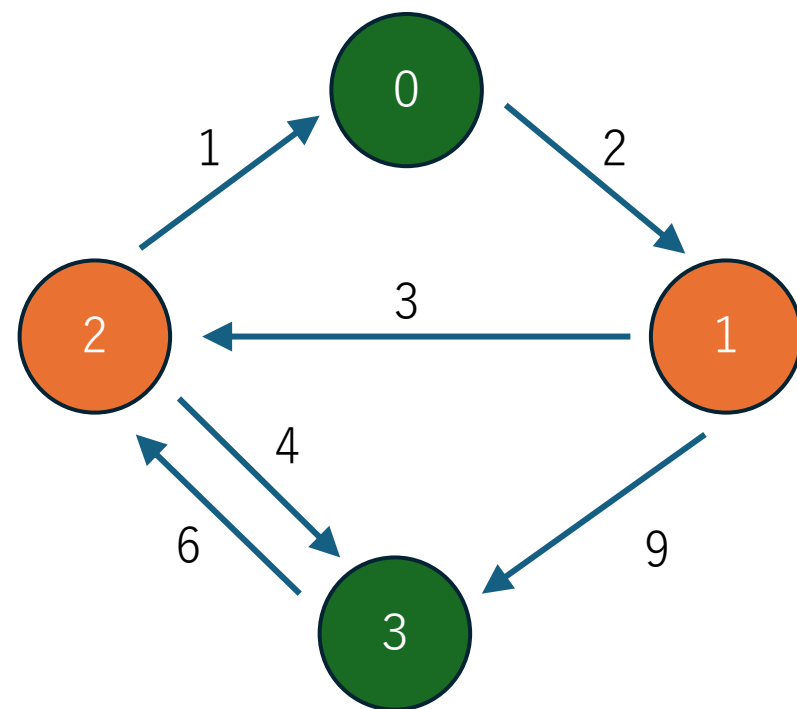
i = 9

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	6
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	9	x	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

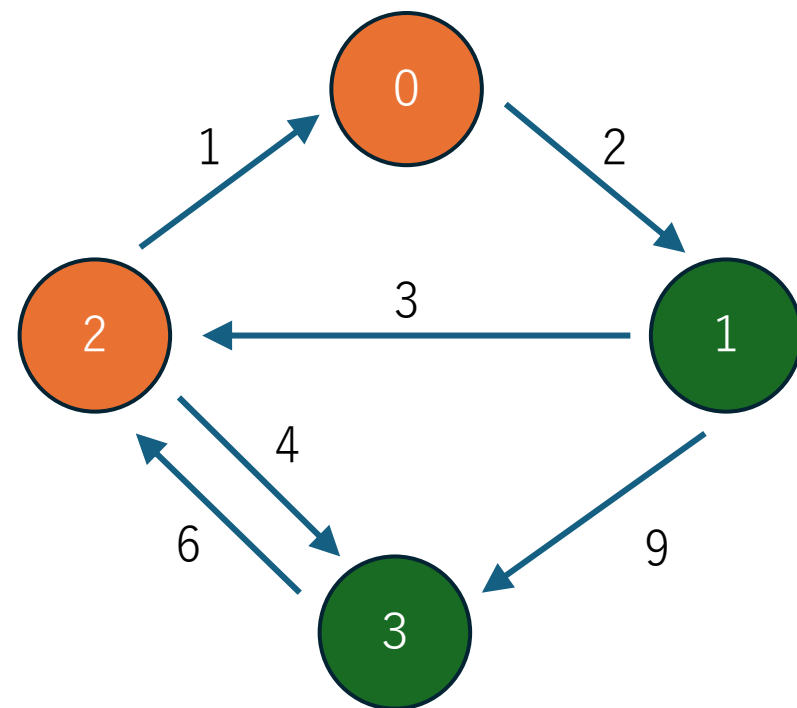
i = 10

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	6
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	9	17	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

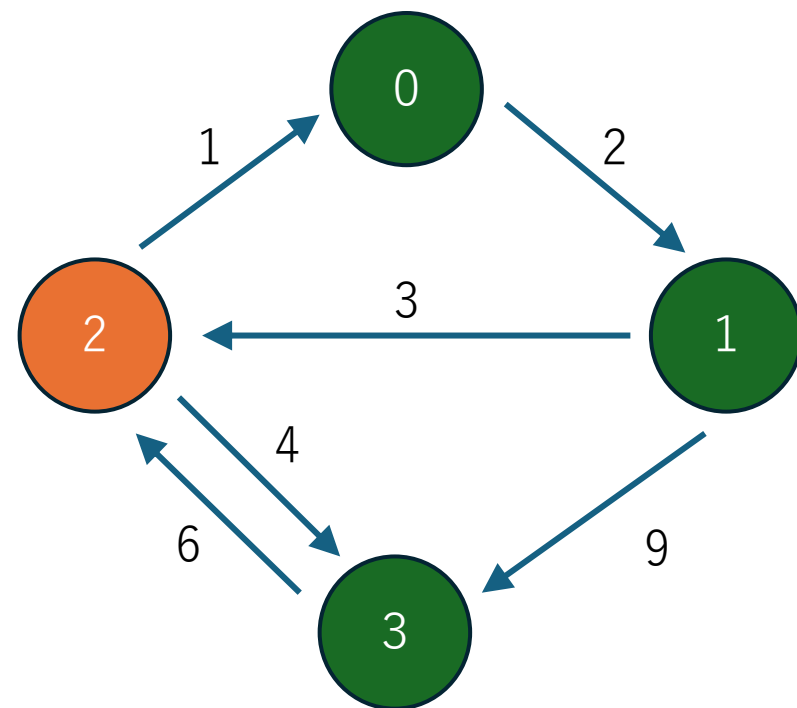
i = 11

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	6
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	9	17	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

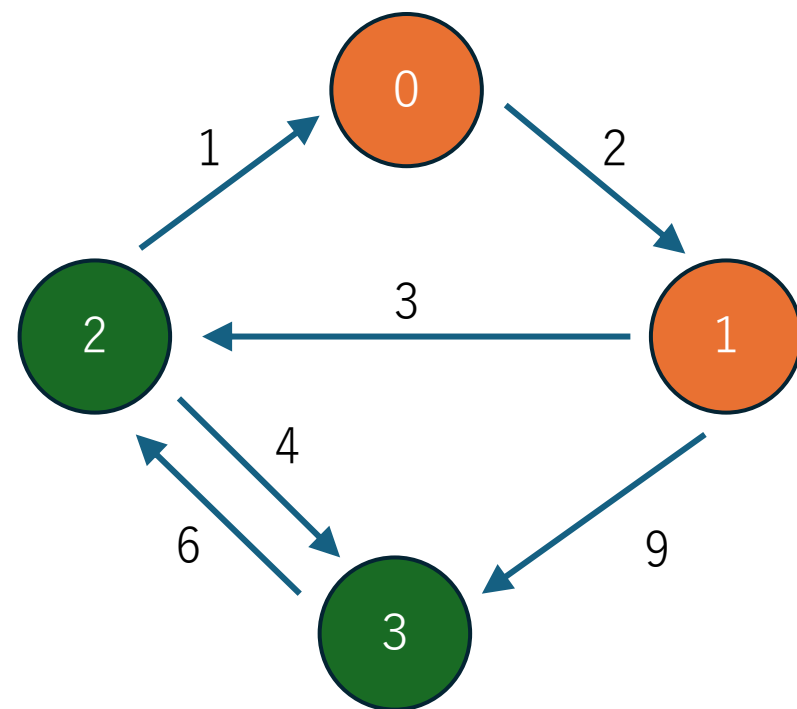
i = 12

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	6
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	9	17	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

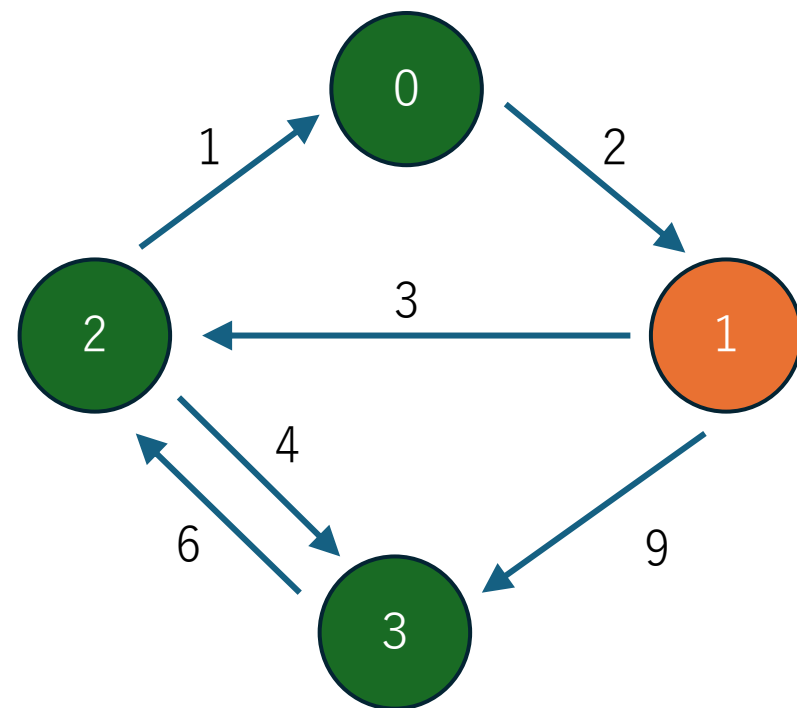
i = 13

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	6
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	9	17	x	x
15(1111)	x	x	x	x

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含めない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

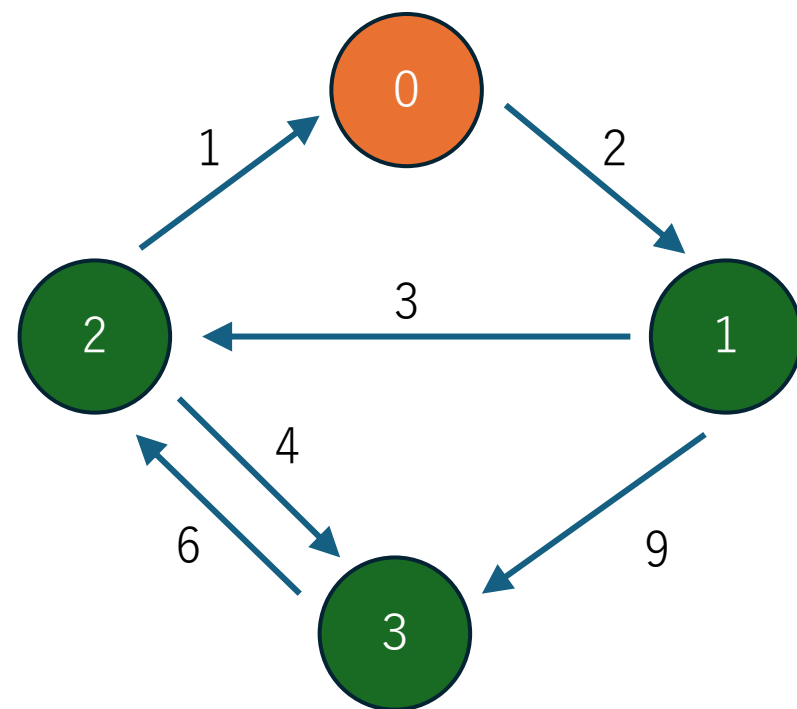
i = 14

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	6
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	9	17	x	x
15(1111)	x	x	x	18

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

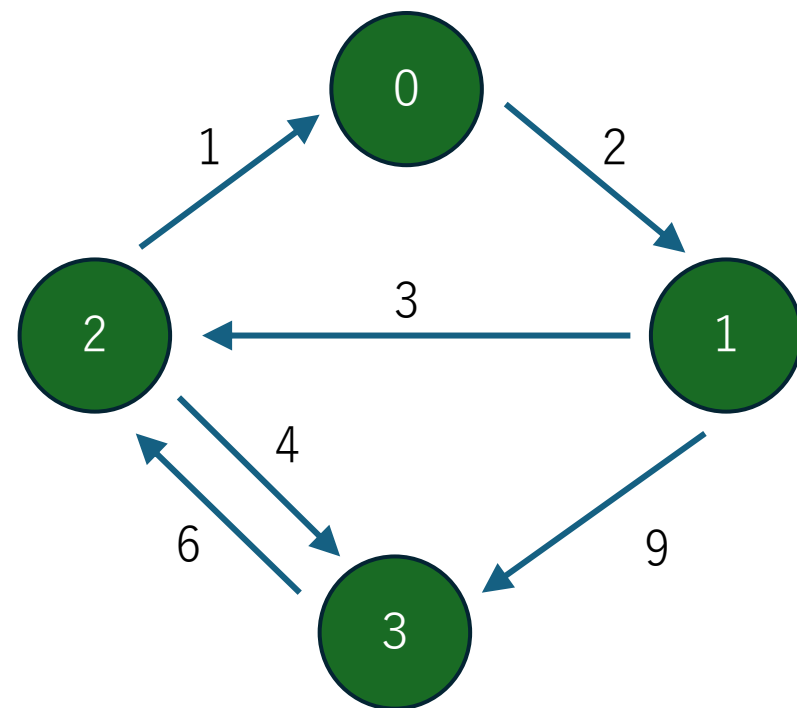
i = 15

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	6
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	9	17	x	x
15(1111)	x	x	x	18

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$



DPの動き

答え

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x
4(0100)	x	x	x	x
5(0101)	x	x	x	x
6(0110)	x	5	x	x
7(0111)	x	x	x	6
8(1000)	x	x	x	x
9(1001)	x	x	x	x
10(1010)	11	x	x	x
11(1011)	x	x	x	x
12(1100)	x	x	x	x
13(1101)	x	x	x	x
14(1110)	9	17	x	x
15(1111)	x	x	x	18

- $dp[i][j]$ = 今までに訪れた頂点の集合が i で、
今 頂点 j にいるとき、考えられる移動した距離の最小値
(最初の頂点0は訪れた頂点に含まない)

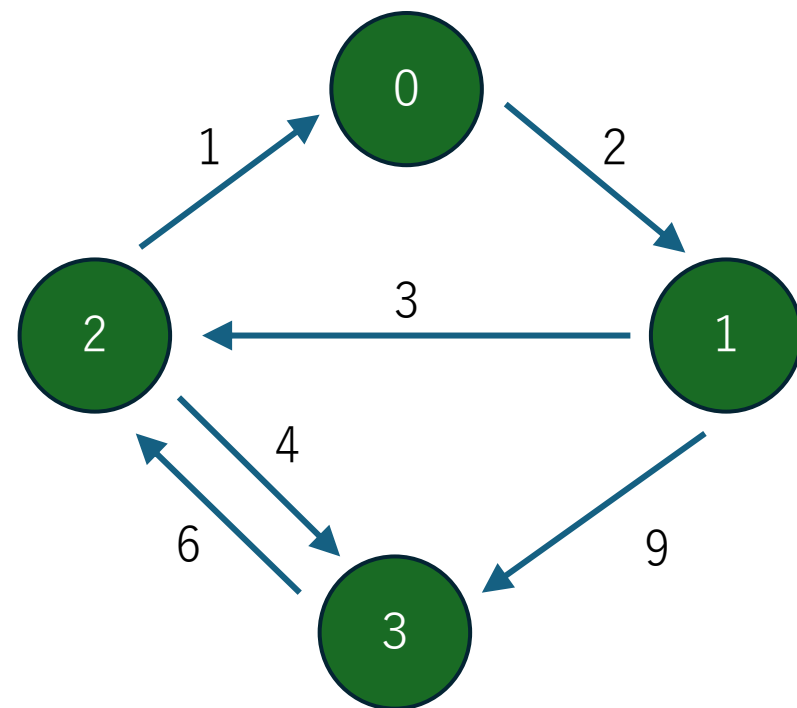
• 遷移

$$dp[i + (1 \ll x)][x] = \min(dp[i + (1 \ll x)][x] , dp[i][j] + |v(j,x)|)$$

答えは

```
if dp[(1<<n)-1][0]がINF => -1
else dp[(1<<n)-1][0]
```

$$dp[15][0] = 18$$



Q. なぜ i を昇順に見て遷移を行えるのか

A. i に対応する集合の真部分集合は i 未満で全て現れるから

	頂点3	頂点 2	頂点1	頂点 0
0 (0000)	x	x	x	0
1(0001)	x	x	x	x
2(0010)	x	x	2	x
3(0011)	x	x	x	x

例えば $3 = \{\text{頂点1}, \text{頂点0}\}$ の真部分集合 $\{\}, \{\text{頂点0}\}, \{\text{頂点1}\}$ はすでに現れている。

i の部分集合からしか i に遷移しないので i を昇順に見るだけでOK！

計算量について

$O(n^2 2^n)$ 程度

```
dp[1<<n][n] = 全ての要素をINFで初期化  
#どこにも訪れてなくて0にいる状態で初期化  
dp[0][0] = 0
```

```
for i = [0..1<<n):  
    for j = [0..n):  
        if dp[i][j]がINFのまま(到達しなかった):continue
```

```
        for (to,cost) in g[j]:  
            if iのtoビット目が1 :continue #すでに通った  
            #遷移  
            dp[i +(1<<to)][to] = min(dp[i + (1<<to)][to],dp[i][j] + cost)
```

答えは

```
if dp[(1<<n)-1][0]がINF => -1  
else dp[(1<<n)-1][0]
```

練習問題

- ABC180E-Traveling Salesman among Aerial Cities

同じような問題です。確認にやってみましょう

- A23-All Free

ちょっと違ったタイプのbitDPです。