Dynamic Programming 2

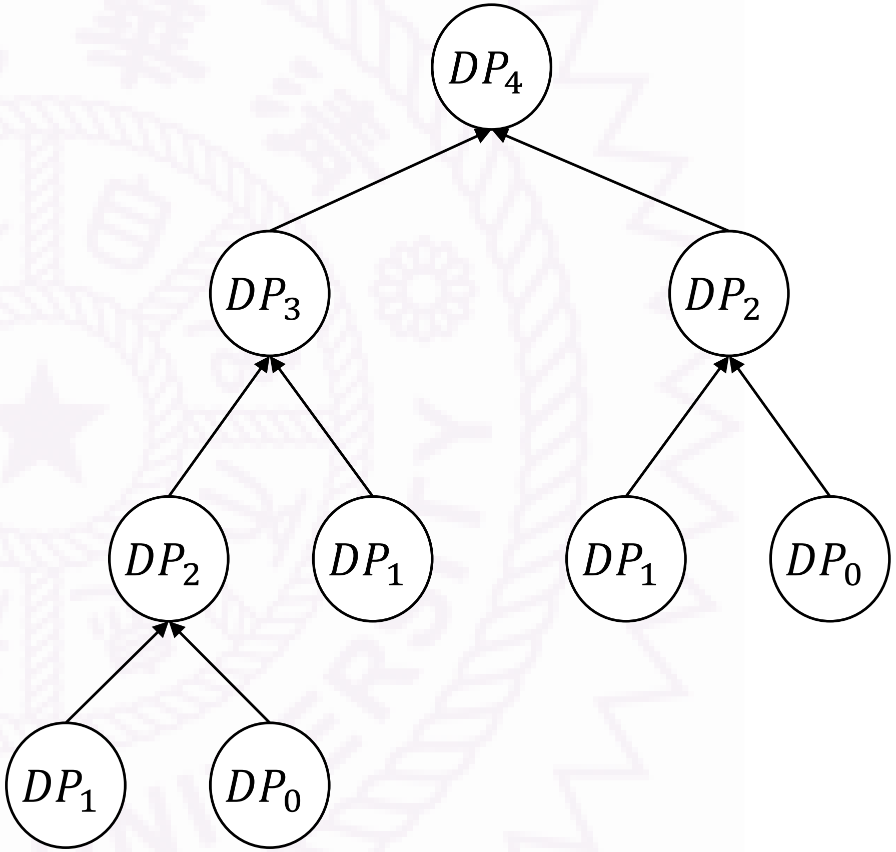
日月卦長

動態規劃
&
有向無環圖

DP & DAG

費式數列

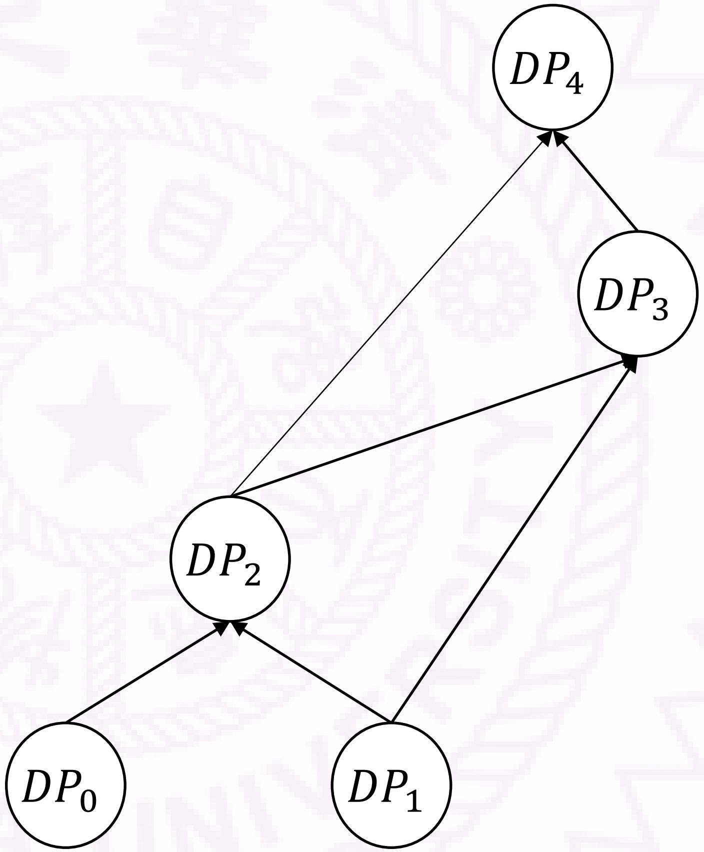
$$DP_n = \begin{cases} 1 & , n \le 1 \\ DP_{n-1} + DP_{n-2} & , n > 1 \end{cases}$$



費式數列

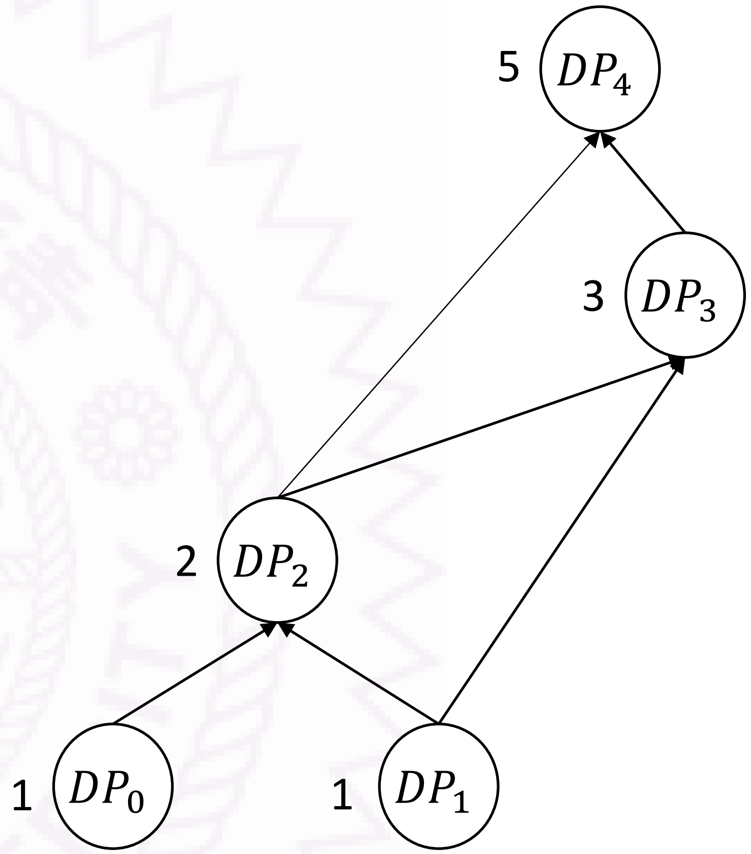
$$DP_n = \begin{cases} 1 & ,n \leq 1 \\ DP_{n-1} + DP_{n-2} & ,n > 1 \end{cases}$$

```
map<int, long long> DP;
long long f(int n) {
  if (n <= 1) return 1;
  if (DP.count(n)) return DP[n];
  return DP[n] = f(n - 1) + f(n - 2);
}
```



DP 與 DAG 的關係

- 將狀態看成點
- 狀態轉移式定義了有向邊
- 會變出一張 DAG
- 狀態的計算順序就是拓樸排序



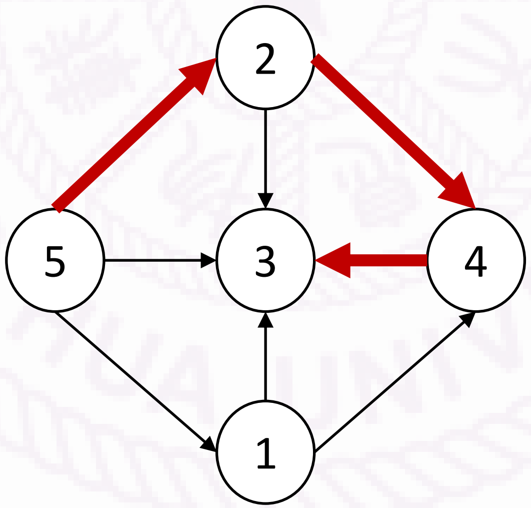
$$DP_n = \begin{cases} 1 & , n \le 1 \\ DP_{n-1} + DP_{n-2} & , n > 1 \end{cases}$$

Atcoder Edu Dp Contest – G. Longest Path

- https://atcoder.jp/contests/dp/tasks/dp_g
- 給你一個有向無環圖，問你最長路徑的長度

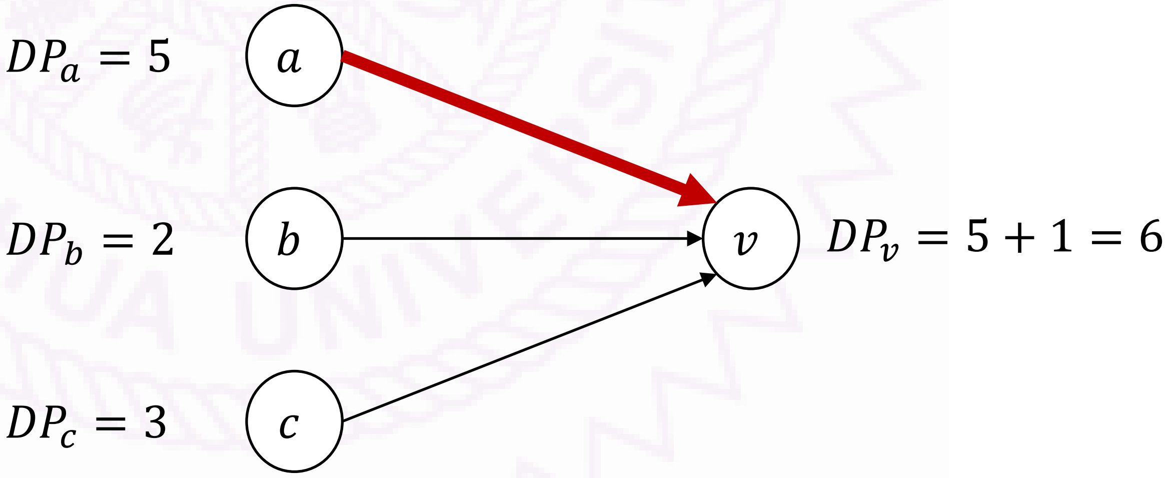
這個問題在一般圖上是 NP complete

5 8
5 3
2 3
2 4
5 2
5 1
1 4
4 3
1 3



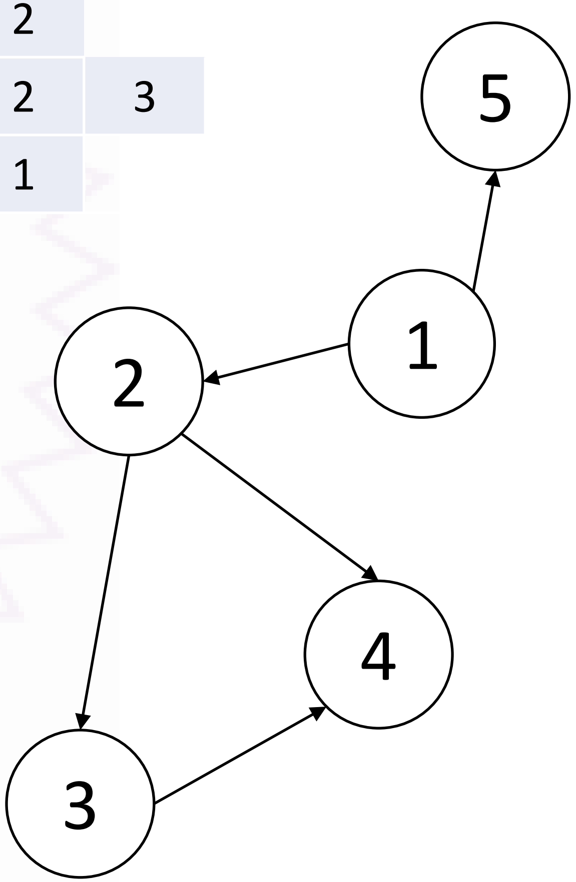
狀態轉移式

$$DP_v = \begin{cases} 0 & , degree^{in}(v) = 0 \\ \max\limits_{(u,v)\in E} \{DP_u\} + 1, & degree^{in}(v) > 0 \end{cases}$$



Top Down – 反著存圖

1		
2	1	
3	2	
4	2	3
5	1	



n 個點, m 條邊

m 條邊

```
5 6
1 2
2 3
2 4
1 5
3 4
```

```cpp
vector<vector<int>> rG;
int n, m;
cin >> n >> m;
rG.assign(n + 1, {});
while (m--) {
   int u, v;
   cin >> u >> v;
   rG[v].emplace_back(u);
}
```

Top Down – 照著公式寫

```
vector<int> DP;
int dfs(int v) {
  if (DP[v] != -1) return DP[v];
  for (int u : rG[v])
    DP[v] = max(DP[v], dfs(u));
  return DP[v] += 1;
}
int solve(int n) {
  DP.assign(rG.size(), -1);
  int ans = 0;
  for (int v = 1; v <= n; ++v)
    ans = max(ans, dfs(v));
  return ans;
}
```

Bottom Up – 紀錄 in-degree

1	2	3	4	5
0	1	1	2	1

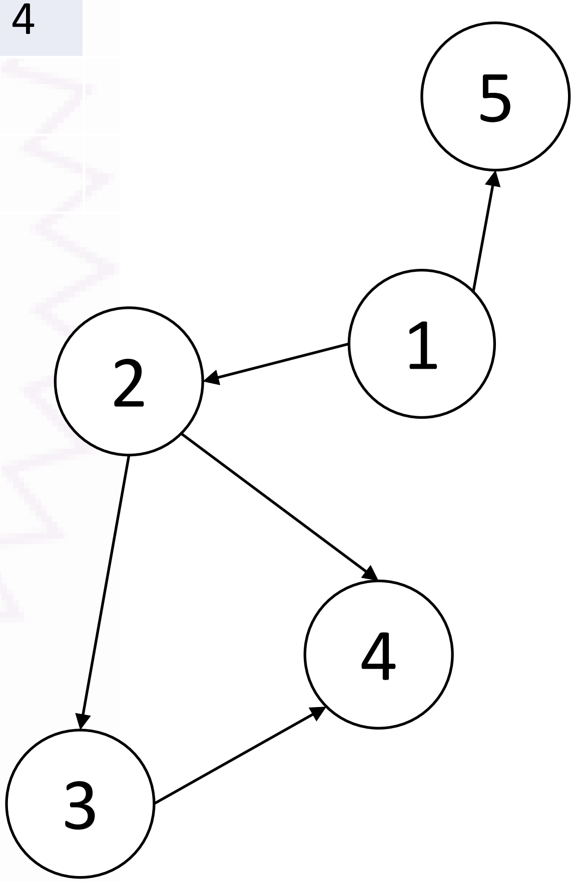
1	2	5
2	3	4
3	4	
4		
5		

n 個點, m 條邊

m 條邊

```
5 6
1 2
2 3
2 4
1 5
3 4
```

```
vector<vector<int>> G;
vector<int> in;
int n, m;
cin >> n >> m;
G.assign(n + 1, {});
in.assign(n + 1, 0);
while (m--) {
  int u, v;
  cin >> u >> v;
  G[u].emplace_back(v);
  ++in[v];
}
```



Bottom Up – 拓樸排序時順便計算

```cpp
int solve(int n) {
  vector<int> DP(G.size(), 0);
  vector<int> Q;
  for (int u = 1; u <= n; ++u)
    if (in[u] == 0)
      Q.emplace_back(u);
  for (size_t i = 0; i < Q.size(); ++i) {
    int u = Q[i];
    for (auto v : G[u]) {
      DP[v] = max(DP[v], DP[u] + 1);
      if (--in[v] == 0)
        Q.emplace_back(v);
    }
  }
  return *max_element(DP.begin(), DP.end());
}
```

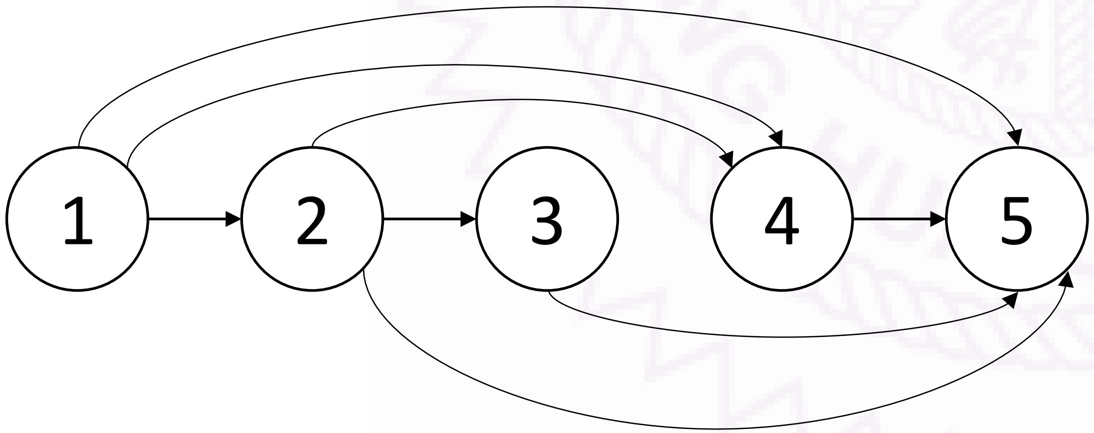
Codeforces Round 988 (Div. 3)
G. Natlan Exploring

- https://codeforces.com/contest/2037/problem/G
- 有 n 個點，編號 $1\sim n$，給你一個序列 $a_1, a_2, \dots, a_n$
- 若 $1 \le i < j \le n$ 且 $\gcd(a_i, a_j) \neq 1$，則點 i 有一條有向邊連到點 j
- 問你點 1 到點 n 總共有幾種路徑

- $1 < n \le 2 \times 10^5$
- $1 < a_i \le 10^6$

範例輸入

- $n = 5$
- $a = [2, 6, 3, 4, 6]$



所有可能路徑

- 1 → 5
- 1 → 2 → 5
- 1 → 2 → 3 → 5
- 1 → 2 → 4 → 5
- 1 → 4 → 5

直觀但是會 TLE 的解

```
#include <bits/stdc++.h>
using namespace std;
constexpr long long MOD = 998244353;
int main() {
  int n;
  cin >> n;
  vector<long long> a(n), dp(n, 0);
  dp[0] = 1;
  for (int i = 0; i < n; ++i) {
    cin >> a[i];
    for (int j = 0; j < i; ++j)
      if (gcd(a[i], a[j]) != 1)
        dp[i] = (dp[i] + dp[j]) % MOD;
  }
  cout << dp[n - 1] << endl;
  return 0;
}
```

這裡算太慢了

排容原理 – 還記得這題嗎?

- 給你兩個正整數 $a, b(1 \le a, b \le 10^9)$
- 問你 $1,2,3, \dots, b$ 中，有多少數和 a 互質

範例 $a = 30, b = 15$

$a = 2 \times 3 \times 5$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

次數
0
1
2
3

- 與 a 不互質的數字數量：

$$\left\lfloor\frac{15}{2}\right\rfloor + \left\lfloor\frac{15}{3}\right\rfloor + \left\lfloor\frac{15}{5}\right\rfloor - \left(\left\lfloor\frac{15}{2 \times 3}\right\rfloor + \left\lfloor\frac{15}{2 \times 5}\right\rfloor + \left\lfloor\frac{15}{3 \times 5}\right\rfloor\right) + \left\lfloor\frac{15}{2 \times 3 \times 5}\right\rfloor = 11$$

$15 - 11 = 4$ 就是與 a 互質的個數

排容原理主要程式碼

```cpp
void inclusion_exclusion_principle(const vector<int> &prime_factor,
                                   function<void(int, int)> callback) {
  long long ans = 0;
  unsigned S = (1u << prime_factor.size()) - 1;
  for (unsigned subset = 1; subset <= S; ++subset) {
    int flag = -1, product = 1;
    for (size_t i = 0; i < prime_factor.size(); ++i) {
      if ((subset >> i) & 1) {
        flag *= -1;
        product *= prime_factor[i];
      }
    }
    callback(flag, product);
  }
}
```

$$dp\_sum[x] = \sum_{1 \le j < i,\ x \mid a_j} dp[j]$$

```
auto get_prime_factors(int N) {
  vector<vector<int>> prime_factors(N + 1);
  for (int i = 2; i <= N; ++i) {
    if (prime_factors[i].empty()) {
      for (int j = i; j <= N; j += i)
        prime_factors[j].emplace_back(i);
    }
  }
  return prime_factors;
}
```

事實上對 a_i 來說不用更新所有的因數
只需要把排容原理用到的那些更新上去就行了

```
auto prime_factors = get_prime_factors(1e6);
vector<long long> dp_sum(1e6 + 1, 0);
auto update = [&](int i) {
  inclusion_exclusion_principle(
      prime_factors[a[i]], [&](int flag, int product) {
        dp_sum[product] = (dp_sum[product] + dp[i]) % MOD;
      });
};
```

對 *dp_sum* 做排容求 *dp*[i]

```
cin >> a[0];
dp[0] = 1;
update(0);
for (int i = 1; i < n; ++i) {
  cin >> a[i];
  inclusion_exclusion_principle(
      prime_factors[a[i]], [&](int flag, int product) {
        dp[i] = (dp[i] + (flag * dp_sum[product] + MOD) % MOD) % MOD;
      });
  update(i);
}
cout << dp[n - 1] << endl;
```

排容原理中的 “flag” – 莫比烏斯函數

- $\mu(n)\begin{cases} 1 & ,n=1 \\ (-1)^k & ,\text{若 } n \text{ 無平方數因數且 } n=p_1p_2\dots p_k \\ 0 & ,\text{otherwise} \end{cases}$

```cpp
auto get_mobius_function(int N) {
  vector<int> mu(N + 1, 1); // Möbius function
  for (int i = 2; i <= N; i++) {
    for (int j = i * 2; j <= N; j += i) {
      mu[j] -= mu[i];
    }
  }
  return mu;
}
```

$$dp\_sum[x] = \sum_{1 \le j < i,\, x \mid a_j} dp[j]$$

```
auto get_factors(int N) {
  vector<vector<int>> factors(N + 1);
  for (int i = 2; i <= N; ++i)
    for (int j = i; j <= N; j += i)
      factors[j].emplace_back(i);
  return factors;
}
```

這裡更新 a_i 的所有因數
不能用在排容的因數 μ 會是 0 所以不影響

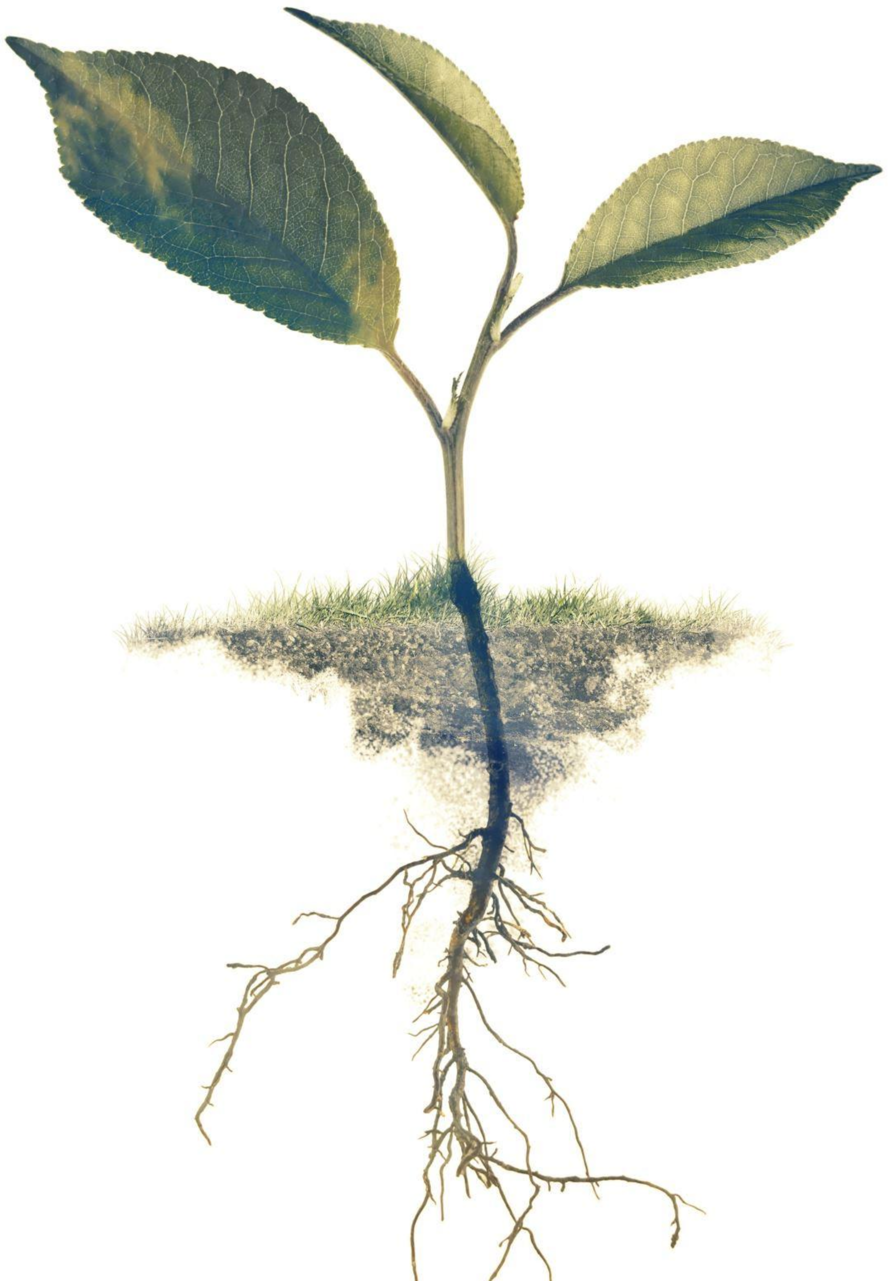
```
auto factors = get_factors(1e6);
vector<long long> dp_sum(1e6 + 1, 0);
auto update = [&](int i) {
  for (int j : factors[a[i]]) {
    dp_sum[j] = (dp_sum[j] + dp[i] + MOD) % MOD;
  }
};
```

對 *dp_sum* 做排容求 *dp*[i]

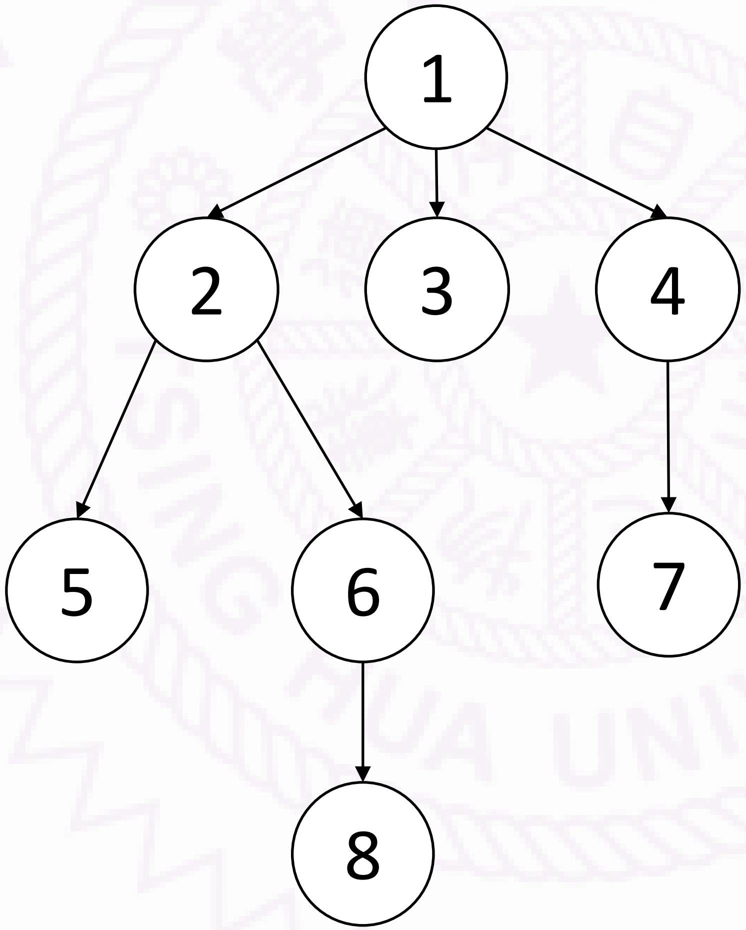
```
auto mu = get_mobius_function(1e6);
cin >> a[0];
dp[0] = 1;
update(0);
for (int i = 1; i < n; i++) {
  cin >> a[i];
  for (int product : factors[a[i]]) {
    dp[i] = (dp[i] + (mu[product] * dp_sum[product] + MOD) % MOD) % MOD;
  }
  update(i);
}
cout << dp[n - 1] << endl;
```

樹上動態規劃

Dynamic Programming on Tree

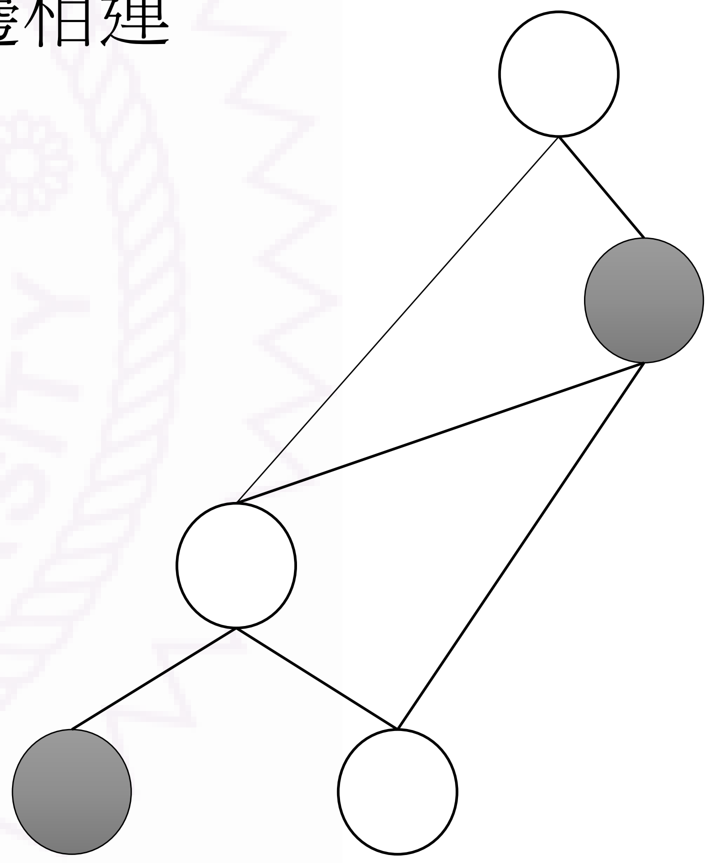


有根樹 ⊂ 有向無環圖



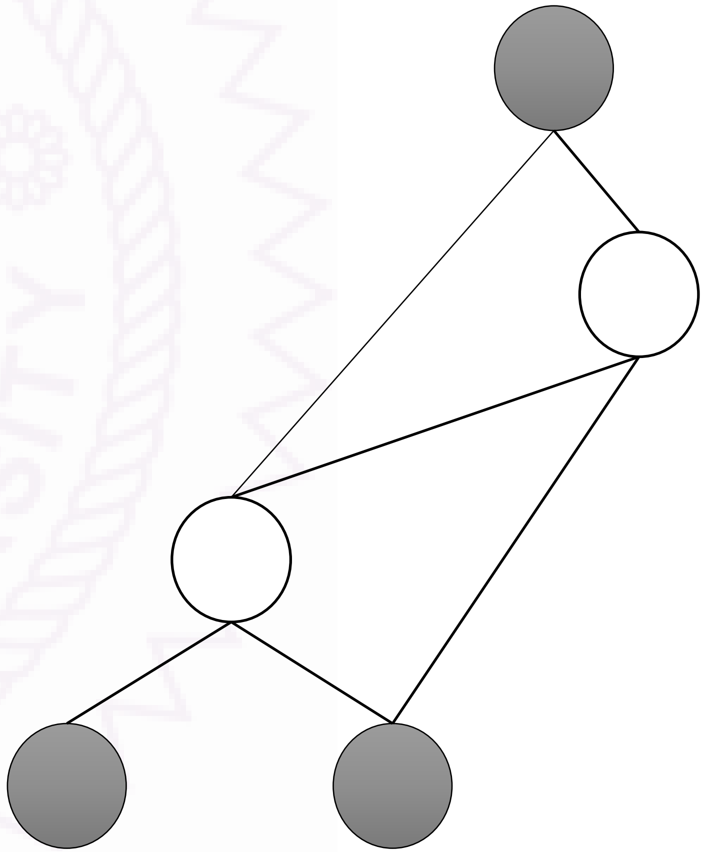
獨立集 Independent set

- 一張圖，選一些點，這些點彼此之間沒有邊相連



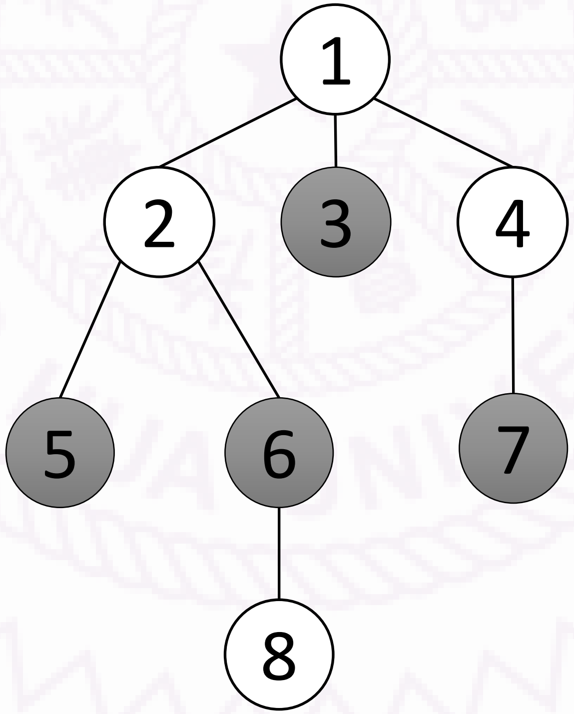
最大獨立集 Maximum independent set

- 一張圖中，點數量最多的獨立集
- 找出最大獨立集在一般圖上是 NP hard
- 但是在樹、二分圖等特殊圖上
  存在高效演算法(方法不同)



樹上最大獨立集

- 輸入一棵有 n 個點的樹，要挑選一群彼此不相鄰的點，而且挑選的點越多點越好。請計算最多可以挑多少點。

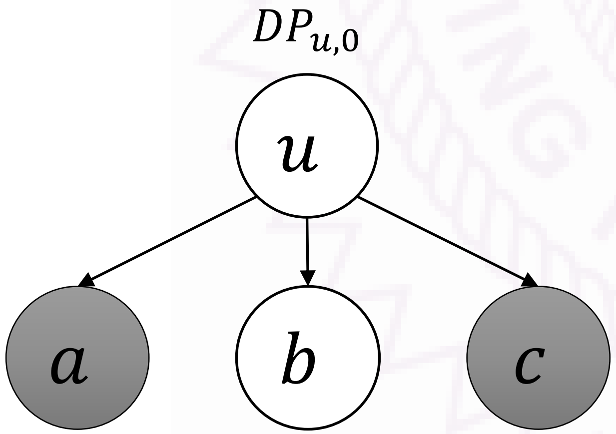


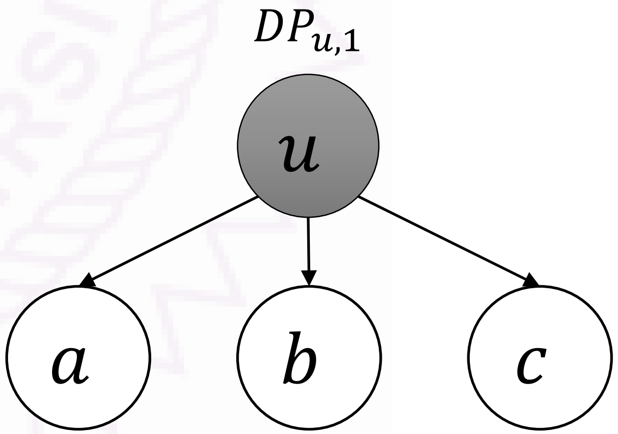
定義狀態

- 假設是有根樹
- $DP_{u,0}$ 表示以 u 為根的子樹，不選 u 時的最佳值
- $DP_{u,1}$ 表示以 u 為根的子樹，選 u 時的最佳值

狀態轉移式

- 不選 u 時， u 的小孩要選不選都是可以的
- 選 u 時， u 的小孩都不能選

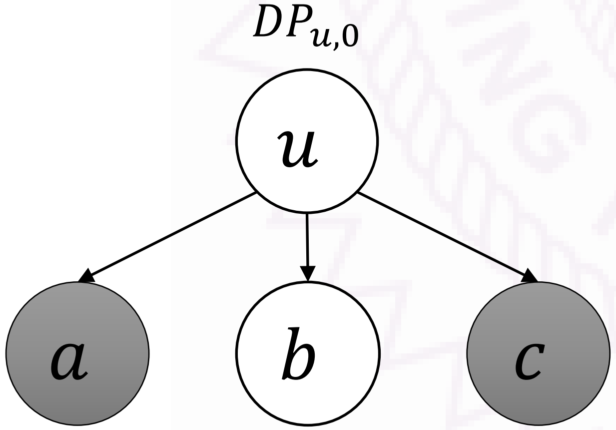


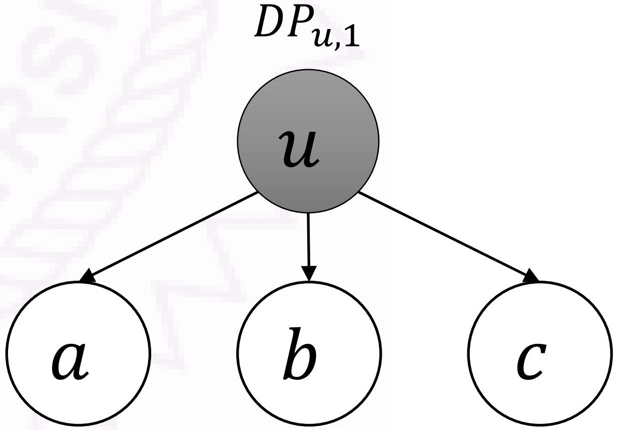


狀態轉移式

$$DP_{u,0} = \sum_{v \in child(u)} \max\{DP_{v,0}, DP_{v,1}\}$$

$$DP_{u,1} = \left(\sum_{v \in child(u)} DP_{v,0} \right) + 1$$





無根樹的輸入
(與圖的輸入相同)

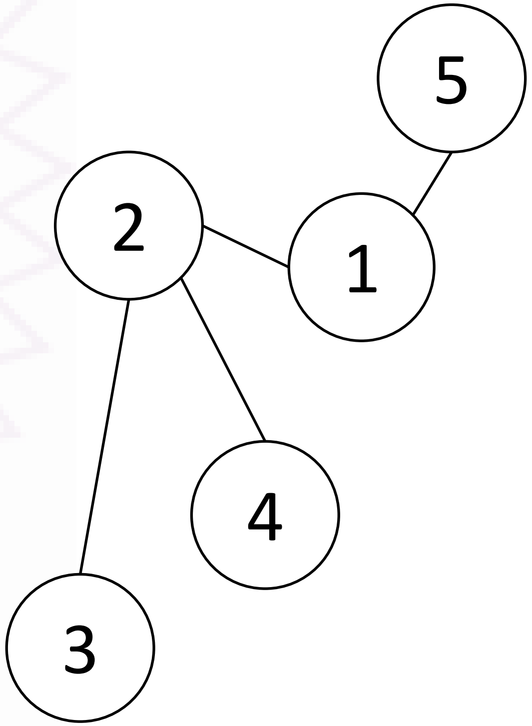
1	2	5	
2	1	3	4
3	2		
4	2		
5	1		

n 個點

```
5
1 2
2 3
2 4
1 5
```

$n-1$ 條邊

```cpp
vector<vector<int>> Tree;
int n;
cin >> n;
Tree.assign(n + 1, {});
for (int i = 0; i < n - 1; ++i) {
  int u, v;
  cin >> u >> v;
  Tree[u].emplace_back(v);
  Tree[v].emplace_back(u);
}
```



程式碼

```
vector<int> DP[2];
int dfs(int u, int pick, int parent = -1) {
  if (u == parent) return 0;
  if (DP[pick][u]) return DP[pick][u];
  if (Tree[u].size() == 1) return pick; // 葉子
  for (auto v : Tree[u]) {
    if (pick == 0) {
      DP[pick][u] += max(dfs(v, 0, u), dfs(v, 1, u));
    } else {
      DP[pick][u] += dfs(v, 0, u);
    }
  }
  return DP[pick][u] += pick;
}
int solve(int n) {
  DP[0] = DP[1] = vector<int>(n + 1, 0);
  return max(dfs(1, 0), dfs(1, 1));
}
```

Travelling salesman problem

旅行推銷員問題

Travelling salesman problem

Travelling salesman problem

示意圖

旅行推銷員問題

- 日日是聖地亞戈集團的推銷員
  他要去美國的 $n(n \leq 15)$ 個城市中推銷金坷垃
  城市的編號為 $0 \sim n-1$

- 設 $dist(x, y)$ 表示城市 x 到城市 y 的距離

- 日日想從聖地亞戈 (城市 0) 出發，經過所有城市恰好各一次後回到聖地亞戈，請你幫助伯爵為日日找出總距離最少的路徑

範例輸入輸出

Input

4

0 10 15 20

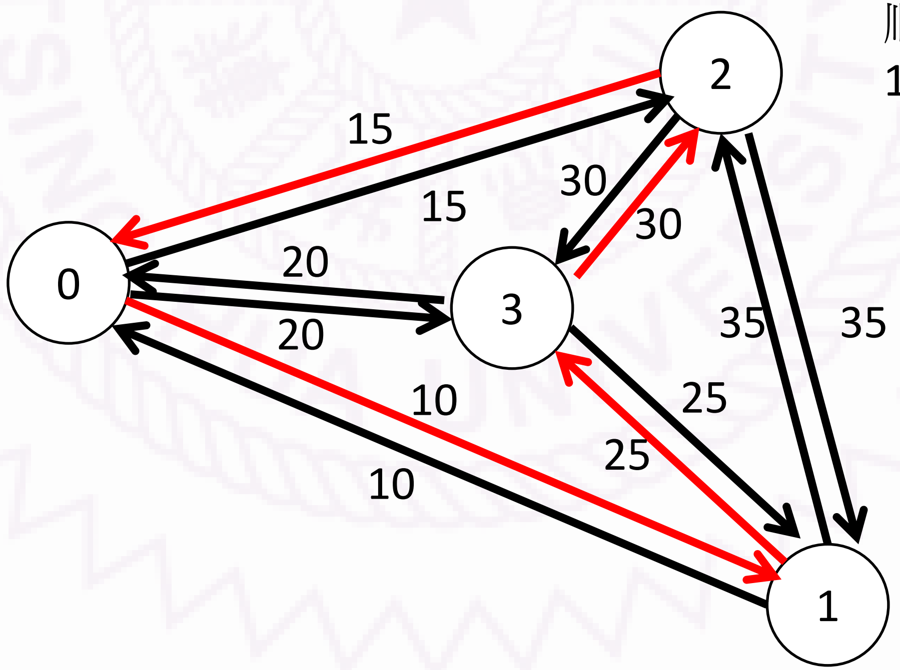
10 0 35 25

15 35 0 30

20 25 30 0

Output

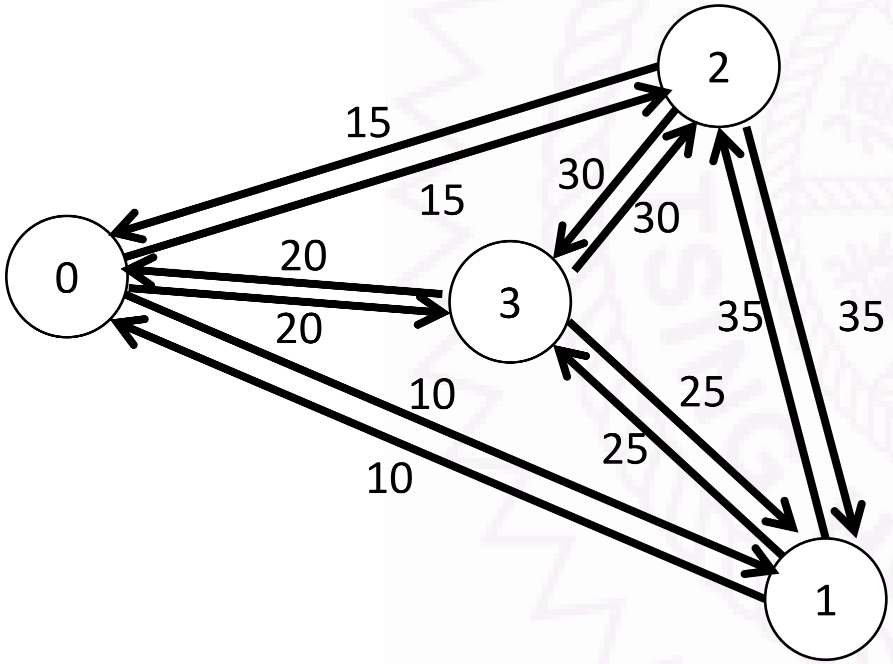
80

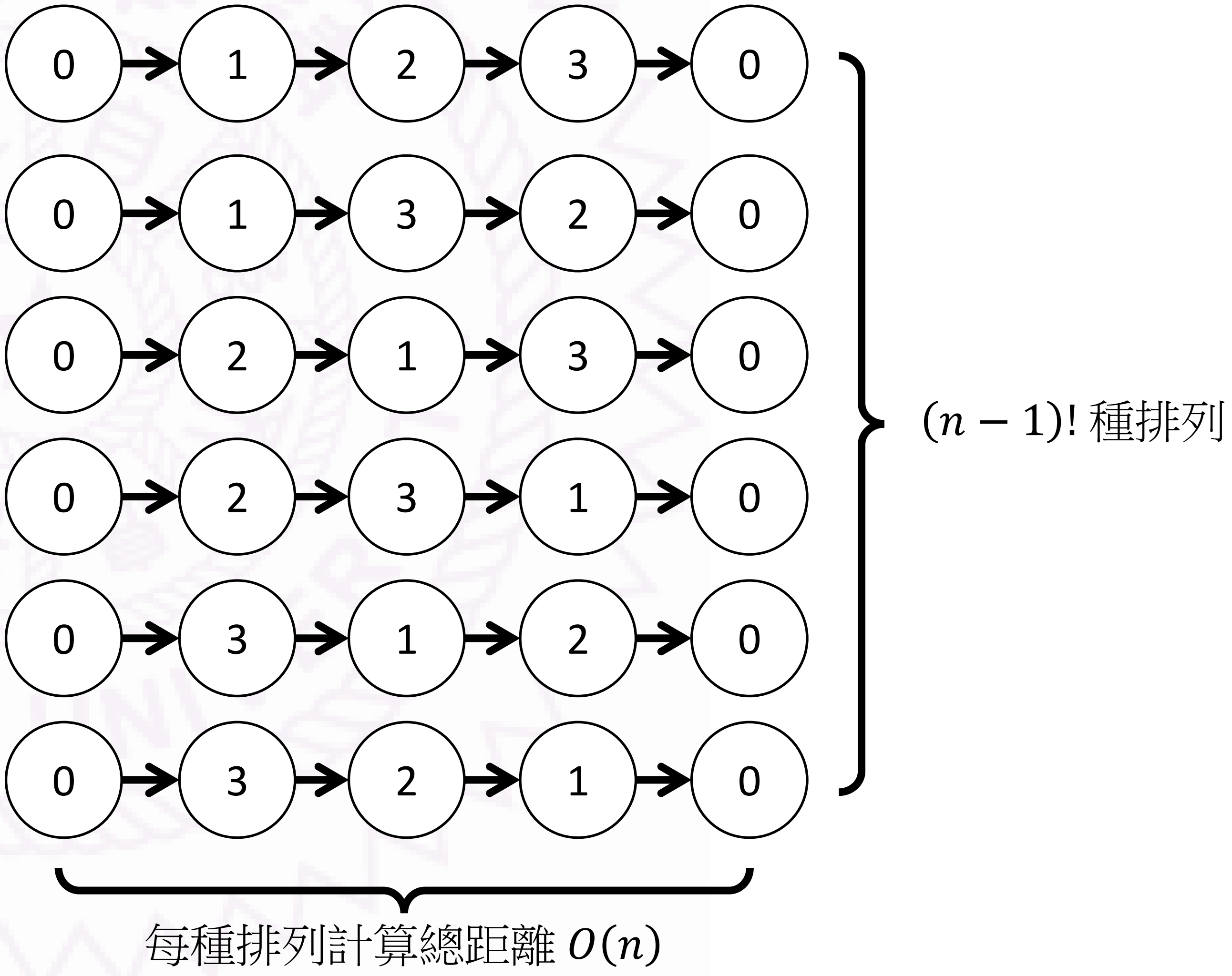


順序：0 -> 1 -> 3 -> 2 -> 0

10+25+30+15 = 80

暴力法 $O(n!)$


0
1
2
3
15
15
30
30
20
20
35
35
10
10
25
25


0 → 1 → 2 → 3 → 0
0 → 1 → 3 → 2 → 0
0 → 2 → 1 → 3 → 0
0 → 2 → 3 → 1 → 0
0 → 3 → 1 → 2 → 0
0 → 3 → 2 → 1 → 0
$(n-1)!$ 種排列
每種排列計算總距離 $O(n)$

暴力法程式碼

```cpp
const int MAXN = 15;
int n; // 點的編號為 0 ~ n-1
int dist[MAXN][MAXN];

vector<bool> used;
int ans;

void dfs(int x, int cost);

int solve() {
  used.resize(n, false);
  ans = 0x3f3f3f3f;
  dfs(0, 0);
  return ans;
}
```

```cpp
void dfs(int x, int cost) {
  bool isAllTrue = true;
  for (auto y : used) isAllTrue &= y;
  if (isAllTrue && x == 0) {
    ans = min(ans, cost);
    return;
  }
  for (int y = 0; y < n; ++y) {
    if (y == x || used[y]) continue;
    used[y] = true;
    dfs(y, cost + dist[x][y]);
    used[y] = false;
  }
}
```

無法成為 DP 的原因

```cpp
const int MAXN = 15;
int n; // 點的編號為 0 ~ n-1
int dist[MAXN][MAXN];

vector<bool> used;
int ans;

void dfs(int x, int cost);

int solve() {
  used.resize(n, false);
  ans = 0x3f3f3f3f;
  dfs(0, 0);
  return ans;
}
```

```cpp
void dfs(int x, int cost) {
  bool isAllTrue = true;
  for (auto y : used) isAllTrue &= y;
  if (isAllTrue && x == 0) {
    ans = min(ans, cost);
    return;
  }
  for (int y = 0; y < n; ++y) {
    if (y == x || used[y]) continue;
    used[y] = true;
    dfs(y, cost + dist[x][y]);
    used[y] = false;
  }
}
```

Step 1: 反向思考讓 ans 變成 local 變數

```cpp
const int MAXN = 15;
int n; // 點的編號為 0 ~ n-1
int dist[MAXN][MAXN];

vector<bool> used;

void dfs(int x);

int solve() {
  used.resize(n, true);
  return dfs(0);
}
```

```cpp
int dfs(int x) {
  bool isAllFalse = true;
  for (auto y : used) isAllFalse &= !y;
  if (isAllFalse) {
    if (x == 0) return 0;
    return 0x3f3f3f3f;
  }
  int ans = 0x3f3f3f3f;
  for (int y = 0; y < n; ++y) {
    if (y == x || !used[y]) continue;
    used[y] = false;
    ans = min(ans, dfs(y) + dist[y][x]);
    used[y] = true;
  }
  return ans;
}
```

Step 2: used 可以直接當成參數

```cpp
const int MAXN = 15;
int n; // 點的編號為 0 ~ n-1
int dist[MAXN][MAXN];

void dfs(int x);

int solve() {
  vector<bool> used(n, true);
  return dfs(0, used);
}
```

```cpp
int dfs(int x, vector<bool> used) {
  bool isAllFalse = true;
  for (auto y : used) isAllFalse &= !y;
  if (isAllFalse) {
    if (x == 0) return 0;
    return 0x3f3f3f3f;
  }
  int ans = 0x3f3f3f3f;
  for (int y = 0; y < n; ++y) {
    if (y == x || !used[y]) continue;
    used[y] = false;
    ans = min(ans, dfs(y, used) + dist[y][x]);
    used[y] = true;
  }
  return ans;
}
```

Step 3: 記憶算過的答案

```cpp
const int MAXN = 15;
int n; // 點的編號為 0 ~ n-1
int dist[MAXN][MAXN];

map<tuple<int, vector<bool>>, int> DP;
void dfs(int x);

int solve() {
  vector<bool> used(n, true);
  return dfs(0, used);
}
```

```cpp
int dfs(int x, vector<bool> used) {
  bool isAllFalse = true;
  for (auto y : used) isAllFalse &= !y;
  if (isAllFalse) {
    if (x == 0) return 0;
    return 0x3f3f3f3f;
  }
  if (DP.count({x, used})) return DP[{x, used}];
  int ans = 0x3f3f3f3f;
  for (int y = 0; y < n; ++y) {
    if (y == x || !used[y]) continue;
    used[y] = false;
    ans = min(ans, dfs(y, used) + dist[y][x]);
    used[y] = true;
  }
  return DP[{x, used}] = ans;
}
```

used 的範圍

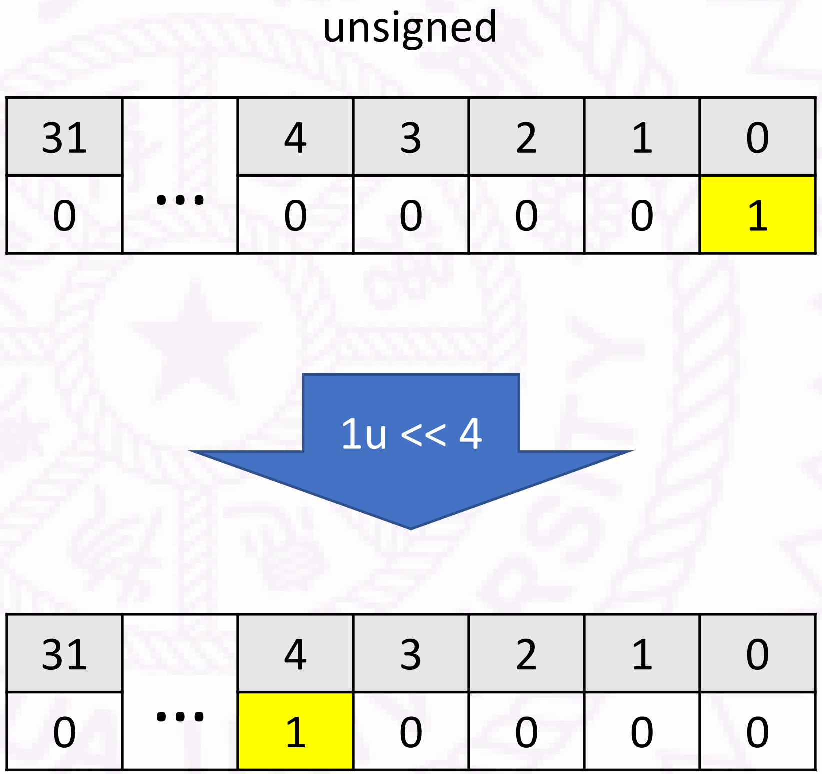
- $n \leq 15$

```
vector<bool> used(n, true);
```

- 把true當成1，false當成0
- 整個 used 可以編碼成 $0 \sim 2^n - 1$ 的正整數
- $2^n \leq 2^{15} = 32768$ 不需要用map存！

Shift

- a << b 表示 $a \times 2^b$
- 1u << 0 = 1
- 1u << 1 = 2
- 1u << 4 = 16



陣列操作➔位元操作

- vector<bool> used(n, true); ➔ unsigned used = (1u << n) - 1;
- if(used[y] == true) ➔ if (used & (1u << y) != 0)
- if(used[y] == false) ➔ if (used & (1u << y) == 0)
- used[y] = !used[y] ➔ used ^= (1u << y)

狀態壓縮 DP (位元 DP)

```
const int MAXN = 15;
int n; // 點的編號為 0 ~ n-1
int dist[MAXN][MAXN];

int DP[MAXN][1u << MAXN];
void dfs(int x);

int solve() {
  return dfs(0, (1u << n) - 1);
}
```

```
int dfs(int x, unsigned used) {
  if (used == 0) {
    if (x == 0) return 0;
    return 0x3f3f3f3f;
  }
  if (DP[x][used]) return DP[x][used];
  int ans = 0x3f3f3f3f;
  for (int y = 0; y < n; ++y) {
    if (y == x || (used & (1u << y)) == 0)
      continue;
    used ^= (1u << y);
    ans = min(ans, dfs(y, used) + dist[y][x]);
    used ^= (1u << y);
  }
  return DP[x][used] = ans;
}
```

時間複雜度

- 狀態數量 $n \times 2^n$

```
int DP[MAXN][1u << MAXN];
```

- 計算每個狀態的時間 $O(n)$

```
for (int y = 0; y < n; ++y)
```

- $O(n^2 2^n)$

迴圈版本

```cpp
int solve() {
  for (unsigned U = 0; U < (1u << n); ++U) {
    for (int x = 0; x < n; ++x) {
      if (U == 0) {
        if (x == 0) DP[x][U] = 0;
        else DP[x][U] = 0x3f3f3f3f;
        continue;
      }
      int ans = 0x3f3f3f3f;
      for (int y = 0; y < n; ++y) {
        if (y == x || (U & (1u << y)) == 0)
          continue;
        U ^= (1u << y);
        ans = min(ans, DP[y][U] + dist[y][x]);
        U ^= (1u << y);
      }
      DP[x][U] = ans;
    }
  }
  return DP[0][(1u << n) - 1];
}
```