自主學習成果報告

競技程式設計演算法

新竹女中 **103** 班

黃惟 **Yui Huang**

個人網站：https://yuihuang.com/

學習成果：

日期	檢定 / 競賽	成績
1/5/2020	APCS 大學程式設計先修檢測	實作五級 (360/400)
2/3/2020	「資訊之芽」入芽考	460 分 (滿分 500)
3/1/2020	Codeforces Round #625 (Div. 2, based on Technocup 2020 Final Round)	Solved: 4 out of 6 (Expert, Rating: 1675)
3/8/2020	TOI 資奧初選	Ranked #89
2/16/2020	AtCoder Beginner Contest 155	Rating 640
4/26/2020	AtCoder Beginner Contest 164	Rating 819 (6 Kyu)
5/10/2020	AtCoder Beginner Contest 167	Rating 829 (6 Kyu)
5/31/2020	AtCoder Beginner Contest 169	Rating 884 (6 Kyu)
6/13/2020	第八屆高一生程式設計排名賽	第 4 名
6/14/2020	AtCoder Beginner Contest 170	Rating 966 (6 Kyu)
6/21/2020	AtCoder Beginner Contest 171	Rating 967 (6 Kyu)

學習綱要：

先備知識：

1 C++ 語法、STL

2 基礎資料結構：

- 2.1 佇列 (queues)【筆記】
- 2.2 堆疊 (stacks)【筆記】
- 2.3 樹狀圖 (tree)，圖形 (graph)【筆記】
  - 2.3.1 BFS (Breadth First Search，廣度優先搜尋)【筆記】
  - 2.3.2 DFS (Depth First Search，深度優先搜索)【筆記】

3 基礎演算法：

- 3.1 排序 (sorting)【筆記】
- 3.2 搜尋 (searching) 【筆記】
- 3.3 貪心法則 (greedy method)【筆記】
- 3.4 動態規劃 (dynamic programming, DP)
  - 3.4.1 零錢問題【筆記】
  - 3.4.2 0-1 背包問題【筆記】
  - 3.4.3 LCS 最長共同子序列【筆記】
  - 3.4.4 LIS 最長遞增子序列【筆記】

§1. Sieve of Eratosthenes 質數篩法

受限文檔大小，圖例請參閱網站：https://yuihuang.com/sieve-of-eratosthenes/

- 【用途】建立質數表，避免費時的除法取餘計算，可供後續程式執行時速查。
- 【實作】
  - 可以改成建立 non-prime table，初始值全部為 false，比較方便。
  - a[ ]：檢查並紀錄給定範圍內的每個數是否為質數。
  - vector<int> v: 紀錄所有已經判斷的質數，供後續程式使用。
  - Line-9：i += j 是關鍵。
- 【範例】ZeroJudge a007: 判斷質數

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
initial	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
0 / 1	F	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
2	F	F	T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T
3	F	F	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T
4 (X)					F																					
5	F	F	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F
6 (X)							F																			
7	F	F	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F

```
1     // 質數篩法 (Sieve of Eratosthenes)
2     bool a[46342];
3     vector <int> v;
4
5     for (int j = 2; j < 46342; j++){
6         if (!a[j]){
7             v.push_back(j);
8
9             for (int i = j * j; i < 46342; i += j){
10                a[i] = true;
11            }
12        }
13    }
```

§2. Euler function 歐拉函數

受限文檔大小，圖例請參閱網站：https://yuihuang.com/euler-function/

- 歐拉函數 Phi(x)：找出「小於或等於 x」 的「正整數」中「與 x 互質」的數的數目。
- 【實作】找出小於或等於 10 的正整數中與 10 互質的數的數目。（參考下圖）
  - 在 [2, sqrt(x)] 區間內，每找到一個 x 的質因數 p，便把 ret (初始值為 x) 乘上 (p-1) / p。（或是 ret -= ret / p）
  - 例如，10 有 2 和 5 兩個質因數，10 * (2 – 1) / 2 = 5，5 * (5 – 1) / 5 = 4。
  - 「小於或等於 10」且「與 10 互質」的數的數目有 4 個：1, 3, 7, 9。
- 【範例】T2498 [SDOI2012] Longge 的问题

```
1    // 歐拉函數 Phi(x)
2    int Phi(int x){
3        int ret = x;
4        int sq = sqrt(x);
5        for (int p=2; p<=sq; p++){
6            if (x % p == 0){
7                while (x % p == 0) x /= p;
8                ret -= ret / p;
9            }
10           if (x == 1) break;
11       }
12       if (x > 1) ret -= ret / x;
13       return ret;
14   }
```

§3. Exponentiating by Squaring 快速冪

受限文檔大小，圖例請參閱網站：https://yuihuang.com/exponentiating-by-squaring/

- 【用途】倍增法：利用「指數」的二進位表示法，來決定計算哪些冪次。
- 【實作】函式 pow_mod(a, b) 計算 a_b % MOD
  - 比如 b = 5 (二進位表示為 101)，這些 bits 按照從右(LSB)到左(MSB)的順序使用。
  - $5 = 2_2 + 2_0$
  - $a_5 = a_{(2^2)} * a_{(2^0)}$
  - Line-13：不斷地乘上自己(squaring)
- 【時間複雜度】
  - 樸素做法：O(n)
  - 快速冪：O(log(n))
- 【練習】Codeforces 1228C. Primes and Multiplication

```
1   ll pow_mod(ll a, ll b){
2       //快速冪 (a^b) % MOD
3       ll ret = 1;
4       ll now = a;
5       while (b) {
6           //b = 5 = b'101
7           //a^b = a^5 = a^4 * a^1
8           if (b & 1) {
9               //需要這個冪次
10              ret *= now;
11              ret %= MOD;
12          }
13          now *= now; //a -> a^2 -> a^4 -> ...
14          now %= MOD; //避免溢位
15          b >>= 1; //b 的二進位數字往右移一位元
16      }
17      return ret;
18  }
```

§4. 數位 DP

- Digit DP，計數用的 DP。
- 【用途】查找 **[0, x] 區間**內，符合條件的個數。用非常少的狀態（處理每個 digit）來得到需要的下一個狀態，避免疊代每個數字衍生的重複計算。
- 【範例】HDU 2089 不要 62【題解】
- dp[7][2][2]：
  - 第一個維度 pos：目前處理的位數。數字 0<n≤m<1000000，最多七位數。
  - 第二個維度 pre：紀錄前一位數是否為 6 (true/false)
  - 第三個維度 lim：是否到達上限 (true/false)。例如 n = 655，假如最高位數是 6 (達上限)，則第二位數最大只能是 5。如果最高位數小於 6 (未達上限)，則第二位數最大可以是 9。
- 函式 dfs( )：從高位數往低位數檢查。
- 函式 solve( )：計算數字的位數，然後呼叫 dfs( )，初始狀態 pre = false, lim = true。
- 題目要計算 [n, m]區間內符合條件的個數，因此答案是 solve(m) – solve(n-1)。
- 【更多練習】Codeforces 1245F. Daniel and Spring Cleaning【題解】

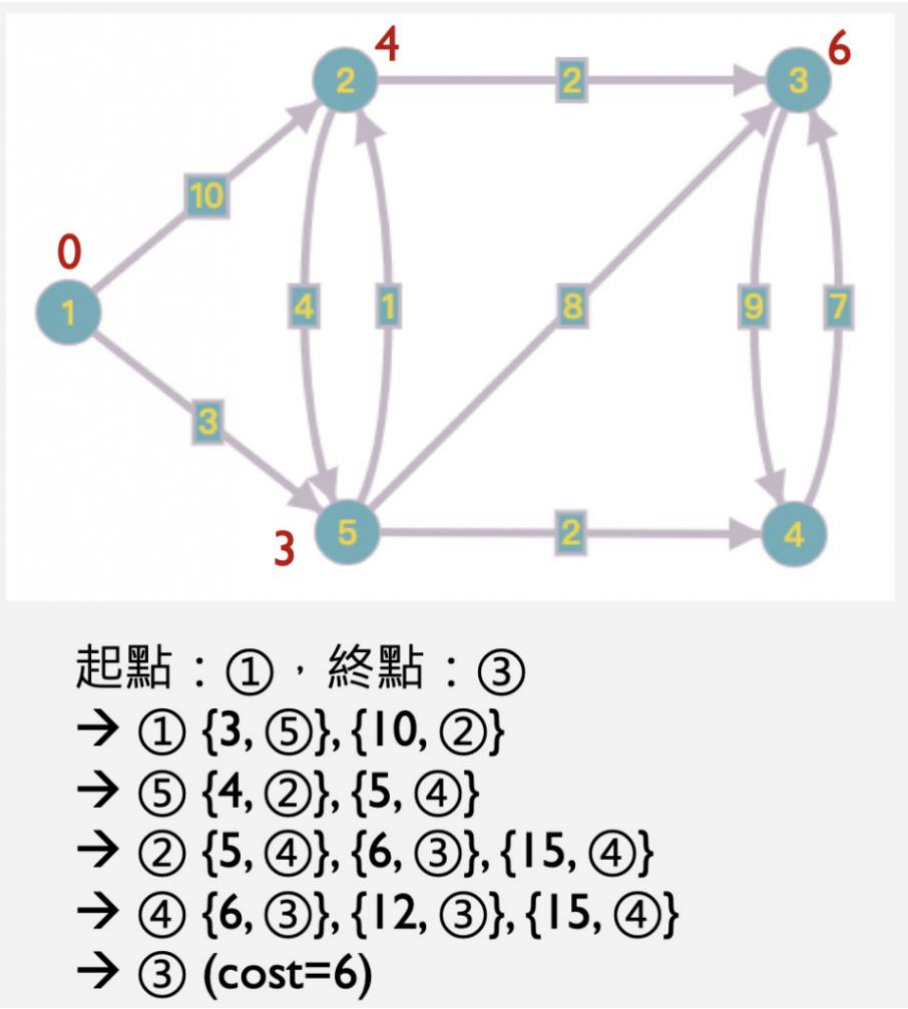
```
#include <iostream>
#include <cstring>
using namespace std;

int n, m, a[7], dp[7][2][2];

int dfs(int pos, int pre, int lim){
    //遞迴終止條件
    if (pos == -1) return 1; //檢查完所有位數，得到一組解
    if (dp[pos][pre][lim] != -1) return dp[pos][pre][lim];
    //ub：這個位數的上限值
    int ub = lim ? a[pos] : 9;
    int ans = 0;
    for (int i = 0; i <= ub; i++){
        if (i == 4) continue; //不能有 4
```

```
16              else if (pre && i == 2) continue; //不要 62
17              ans += dfs(pos-1, i==6, lim && i==a[pos]);
18          }
19          dp[pos][pre][lim] = ans;
20          return ans;
21      }
22
23      int solve(int x){
24          int cnt = 0;
25          memset(dp, -1, sizeof(dp));
26          //cnt：數字 x 有幾位數
27          while (x){
28              a[cnt] = x % 10;
29              x /= 10;
30              cnt++;
31          }
32          return dfs(cnt-1, 0, 1);
33      }
34
35      int main() {
36          while (cin >> n >> m){
37              if (n == 0 && m == 0) break;
38              cout << solve(m) - solve(n-1) << "\n";
39          }
40      }
```

§5. Dijkstra algorithm 單點源最短路徑

受限文檔大小，圖例請參閱網站：https://yuihuang.com/dijkstra-algorithm/

- 【用途】給定一張有向圖，找出起點與終點(或其它頂點)之間的最短路徑。
- 【條件】沒有負權重的邊。
- 【原理】Greedy + DP
  - 如果存在一條邊 【s –> t】，權重為 w，則從【起點 –> t】的最短路徑可以透過【起點 –> s】+ 【s –> t】 來拓展。
  - 每次都利用已經確定最短距離的頂點來拓展到其相鄰的頂點。
  - 如果 dis[s] + w 比 dis[t] 小，則更新【起點 –> t】的距離成 dis[t] = dis[s] + w。
  - 持續拓展每條邊，直到所有的 dis[x] 都代表【起點 –> x】的最短距離。
  - 每條邊只會被拓展一次。
- 【實作】
  - vector<pair<int, int> >：存圖。first 是「下一個頂點」的編號，second 是連結「下一個頂點」的邊的權重，。
  - 陣列 dis[ ]：用來更新起點到各頂點的最短距離。
  - #define P pair<long long, int>
  - 【方法 1】priority_queue<P, vector<P>, **greater**<P> > 儲存尚未拓展的頂點，first 是起點到「鄰接頂點」的最短距離，second 是「鄰接頂點」的編號。使用 greater 來對 first 進行由小到大的排序。
  - 【方法 2】set<P> 儲存尚未拓展的頂點，first 是起點到「鄰接頂點」的最短距離，second 是「鄰接頂點」的編號。 (st.erase 可縮減查找的時間。)
  - Line-19：避免走訪已經處理過的頂點。
- 【時間複雜度】O(E * log (V))，其中 V 為圖的頂點個數，E 是邊數。
- 【範例】UVa 10986 – Sending email



【方法 1】priority_queue

```
1     // Dijkstra - priority_queue
2     memset(dist, 0x3F, sizeof(dist));
3     priority_queue<P, vector<P>, greater<P> > pq;
4     dist[S] = 0;
5     pq.push({0, S});
6     while (!pq.empty()){
7         P p = pq.top();
8         pq.pop();
9         if (dist[p.S] < p.F) continue;
10        for (auto nxt: G[p.S]){
11            if (dist[nxt.F] > dist[p.S] + nxt.S){
12                dist[nxt.F] = dist[p.S] + nxt.S;
```

```
13                  pq.push({dist[nxt.F], nxt.F});
14              }
15          }
16      }
```

【方法 2】set

```
1       // Dijkstra - set
2       memset(dist, 0x3F, sizeof(dist));
3       set<P> st;
4       dist[S] = 0;
5       st.insert({0, S});
6       while (!st.empty()){
7           P p = *st.begin();
8           st.erase(p);
9           if (dist[p.S] < p.F) continue;
10          for (auto nxt : G[p.S]){
11              long long new_dist = p.F + nxt.S;
12              if (new_dist < dist[nxt.F]){
13                  //st.erase({dist[nxt.F], nxt.F});
14                  dist[nxt.F] = new_dist;
15                  st.insert({dist[nxt.F], nxt.F});
16              }
17          }
18      }
```

§6. Floyd-Warshall algorithm 全點對最短路徑

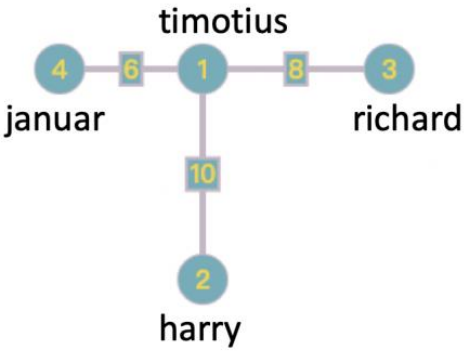
受限文檔大小，圖例請參閱網站：https://yuihuang.com/floyd-warshall-algorithm/

- 【用途】用來解決「有向圖」中，任意兩點間的最短路徑。可以正確處理有「負權」的邊。
- 【原理】枚舉 + DP
- 【實作】
  - 枚舉所有中間點 (i)，更新所有【j –> k】的最短路徑。
  - dis[j][k] = dis[j][i] + dis[i][k]
- 【複雜度】
  - 時間複雜度：$O(N_3)$
  - 空間複雜度：$O(N_2)$
- 【範例】ZeroJudge d282: 11015 – 05-2 Rendezvous
- 【更多練習】TIOJ 1034．搶救雷恩大兵 (Saving Ryan)

```
1   // 初始值：起點終點相同時設為 0，其餘設為 INF
2   // 因為後續有加法運算，因此 INF = 0x3FFFFFFF
3   for (int i = 0; i < maxn; i++){
4       for (int j = 0; j < maxn; j++){
5           if (i == j) dis[i][j] = 0;
6           else dis[i][j] = 0x3FFFFFFF;
7       }
8   }
```

```
1    // Floyd-Warshall 做法
2    // i：枚舉中間點
3    // j, k：計算 all-pairs shortest path
4    for (int i = 0; i < maxn; i++){
5        for (int j = 0; j < maxn; j++){
6            for (int k = 0; k < maxn; k++){
7                if (dis[j][k] > dis[j][i] + dis[i][k]){
```

```
8                       dis[j][k] = dis[j][i] + dis[i][k];
9                   }
10              }
11          }
12      }
```



Initial

	1	2	3	4
1	0	∞	∞	∞
2	∞	0	∞	∞
3	∞	∞	0	∞
4	∞	∞	∞	0

Input data

	1	2	3	4
1	0	10	8	6
2	10	0	∞	∞
3	8	∞	0	∞
4	6	∞	∞	0

枚舉中間點 = 1

	1	2	3	4
1	0	10	8	6
2	10	0	18	16
3	8	18	0	14
4	6	16	14	0

依序枚舉中間點 = 2, 3, 4

	1	2	3	4
1	0	10	8	6
2	10	0	18	16
3	8	18	0	14
4	6	16	14	0

結果

	1	2	3	4	Sum
1	0	10	8	6	24
2	10	0	18	16	44
3	8	18	0	14	40
4	6	16	14	0	36

§7. Topological Sort 拓撲排序

受限文檔大小，圖例請參閱網站：https://yuihuang.com/topological-sort/

- 【用途】有向無環圖(Directed Acyclic Graph, DAG)進行順序排序。
  - 圖形的「頂點」表示任務編號。
  - 圖形的「邊」表示一個任務是另一個任務的前置作業。
  - 拓撲排序即為一個有效的任務執行順序。
- 【實作】
  - vector<int> g：每個頂點「出邊」連結的頂點。
  - in[ ]：紀錄每個頂點的 in-degree（「入邊」數目）
  - queue<int> q：紀錄 in-degree 為零的頂點
  - vector<int> ans：紀錄有效的拓撲順序
  - 最後檢查 ans 的長度是否等於頂點的個數
- 【時間複雜度】
  - O(V + E)，其中 V 為圖的頂點個數，E 是邊數。
- 【範例】TIOJ 1092 . A.跳格子遊戲
- 【更多練習】Codeforces 510C. Fox And Names

```
1    for (int i=0; i<m; i++){
2        cin >> a >> b;
3        //將頂點編號轉成 zero-based
4        a--;
5        b--;
6        g[a].push_back(b);
7        in[b]++;
8    }
```

```
1    queue<int> q;
2    for (int i=0; i<n; i++){
3        //找出「入邊」(in-degree)為零的頂點
```

```
4         if(in[i] == 0) q.push(i);
5     }
```

```
1     vector<int> ans;
2     while(!q.empty()){
3         int now = q.front();
4         q.pop();
5         ans.push_back(now);
6
7         //遍歷子任務
8         for(auto nxt: g[now]){
9             in[nxt]--;
10            if(in[nxt] == 0) q.push(nxt);
11        }
12    }
```

§8. 並查集

Disjoint Set Union-find algorithm (DSU)

受限文檔大小，圖例請參閱網站：https://yuihuang.com/union-find-algorithm/

- 【用途】並查集是一種資料結構，用於處理不相交集合(Disjoint Sets)的合併及查詢。操作時使用 Union-Find algorithm。
- 【原理】
  - 資料結構：有向圖
  - 每個點有一個出邊(out-degree) 代表每個頂點所屬集合的「代表元素」，初始值指向自己。
  - 查詢時如果遇到指向自己的點 x，則 x 即為該集合的代表元素，回傳 x，否則繼續查詢。
- 【實作 1】
  - 初始化陣列 p[i] = i，表示指向自己。
  - Find( )：找出目標元素屬於哪一個集合 。【版本 2】 順便進行路徑壓縮(邊的縮減)，可縮短後續查詢的時間。
  - Union( )：把兩個集合合併成一個。
- 【實作 2】 可連帶查詢集合的成員數目
  - 初始化陣列 p[i] = -1，表示指向自己。
  - p[代表元素] = 集合內的成員數目(負數)。
  - 當 i 不是所屬集合的代表元素時，p[i] = 代表元素編號。
- 【範例】ZeroJudge d831: 畢業旅行
- 【練習】ZeroJudge d449: 垃圾信件

【實作 1】

```
1     //版本-1
2     int Find(int x){
3         // 當 p[x] = x，x 即為該所屬集合的代表元素
4         // 否則遞迴繼續找
5         return p[x] == x ? x : find(p[x]);
```

```
6       }
7
8       //版本-2: 路徑壓縮
9       int Find(int x){
10          // 當 p[x] = x，x 即為該所屬集合的代表元素
11          // 否則遞迴繼續找，順便做「路徑壓縮」
12          return p[x] == x ? x : p[x] = Find(p[x])
13      }
```

```
1   void Union(int x,int y){
2       int g1 = Find(x); //找出 x 所屬集合的代表元素
3       int g2 = Find(y); //找出 y 所屬集合的代表元素
4       if(g1 != g2) p[g2] = g1; //將 g2 指向 g1，即為合併兩個集合，g1 為代表元素
5   }
```

【實作 2】可連帶查詢集合的成員數目

```
1   int Find(int x){
2       // 初始值 p[x] = -1 表示指向自己
3       return p[x] < 0 ? x : p[x] = find(p[x]);
4   }
```

```
1   void Union(int x, int y){
2       int g1 = Find(x);
3       int g2 = Find(y);
4       if(g1 != g2){
5           p[g1] += p[g2]; //加總元素個數(負值)
6           p[g2] = g1; //p[]:只有代表元素的值是存元素個數，其它的存代表元素編號
7       }
8   }
```

§9. 最小生成樹
Minimum Spanning Tree (MST)

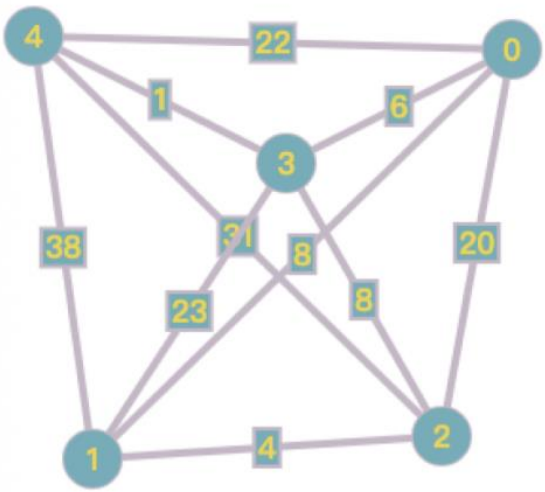
受限文檔大小，圖例請參閱網站：https://yuihuang.com/minimum-spanning-tree/

- 【用途】從一張有權無向圖中 (N 個頂點)，選出一些邊，形成一棵包含所有頂點的樹，條件是所選的邊的權值總和要最小。
- 【原理】
  - 利用並查集的特性，當選擇一條邊後，對應兩個頂點的集合即可合併成同一個集合。
  - 當最後只剩下一個集合或已經使用 N-1 條邊時，便構成一棵樹。
- 【實作】
  - 排序：先把所有邊依照權值由小到大排序。
  - 疊代：依序疊代每條邊，若該邊的兩個頂點屬於不同集合，即採用這條邊，並合併兩點所屬集合。
  - Union-Find algorithm 同並查集。
  - 最後把所有被選到的邊的權值加起來，即為構成最小生成樹的成本。
- 【範例】ZeroJudge d909: 公路局長好煩惱！？

```
1    //定義一結構存放每一條邊的資料
2    //x, y 為兩個頂點，c 為這條邊的權值
3    struct Edge {
4        int x, y;
5        int cost;
6    };
```

```
1     vector <Edge> v; //存放邊的資料
2     int todo = n-1; //n 個節點的樹，有 n-1 條邊
3     int mst_cost = 0;
```

```
4
5       //針對 cost 排序，由小排到大
6       sort(v.begin(), v.end(), cmp);
7
8       //disjoint sets, Union-Find algorithm
9       for (Edge edge:v){
10          int g1 = find(edge.x);
11          int g2 = find(edge.y);
12          if (g1 == g2) continue;
13          else {
14              p[g2] = g1;
15              todo -= 1;
16              mst_cost += edge.cost;
17          }
18      }
```



Initial
(todo = 4, mst_cost = 0)

	0	1	2	3	4
p[i]	0	1	2	3	4

{3, 4, 1}
(todo = 3, mst_cost = 1)

	0	1	2	3	4
p[i]	0	1	2	3	3

{1, 2, 4}
(todo = 2, mst_cost = 5)

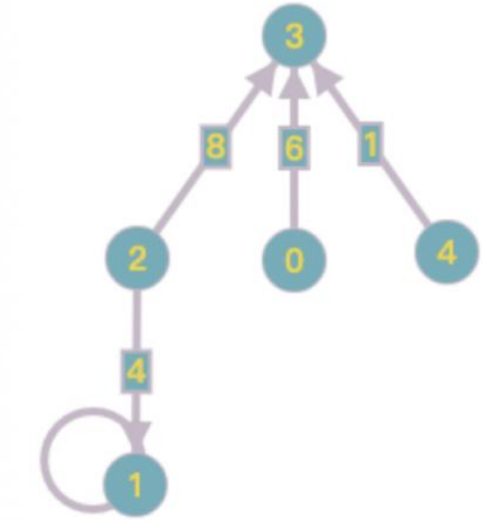
	0	1	2	3	4
p[i]	0	1	1	3	3

{3, 0, 6}
(todo = 1, mst_cost = 11)

	0	1	2	3	4
p[i]	3	1	1	3	3

{3, 2, 8}
(todo = 0, mst_cost = 19)

	0	1	2	3	4
p[i]	3	1	3	3	3



§10. 最近共同祖先

Lowest Common Ancestor

受限文檔大小，圖例請參閱網站：https://yuihuang.com/lowest-common-ancestor/

- 【用途】找出樹上兩點(x 、 y)的最短距離，可以從 x 先往上走到層數最深的共同祖先 (最靠近自己的祖先)，接著在往下走到 y。
  - 根節點的層數為 1。
  - 下圖中的節點 3 和 4 的最近共同祖先(LCA) 為節點 1。
- 【實作】

- 從根節點開始進行 DFS 遍歷整棵樹，可以求得每個節點的層數 (depth) 及其子樹包含的節點個數 (size)。
- p[j][i]：parent 陣列，紀錄「節點-j」的第 2^i 倍祖先。
  - p[j][0]：「節點-j」的 parent。
  - p[j][i] = p[p[j][i-1]] [i-1]
- lca(x, y)：求兩點的 LCA（利用倍增法）
  - 先利用 parent 陣列把較深的點移至和較淺的點「同層」。
  - 若兩點相等，即為答案；
  - 若兩點不相等，利用一個 while 迴圈，每次兩個點都跳到各自的 parent，直到兩點相等。
- 【範例】Kattis – Boxes

```
//從根節點 DFS 遍歷整棵樹，可以求得每個節點的層數 (depth)
//及其子樹包含的節點個數 (size)。
int dfs(int node, int d){
    dep[node] = d + 1;
    if (v[node].empty()){
        siz[node] = 1;
        return 1;
    }
    int total = 1;
    for (int i:v[node]){
        total += dfs(i, d+1);
    }
    siz[node] = total;
    return siz[node];
}
```

```
// 找出每個節點的 2^i 倍祖先
// 2^20 = 1e6 > 200000
for (int i = 1; i < 20; i++){
    for (int j = 1; j <= n; j++){
        p[j][i] = p[p[j][i-1]][i-1];
    }
```

```
7     }
```

```
1     // 求兩點的 LCA（利用倍增法）
2     int lca(int a, int b){
3         if (dep[b] < dep[a]) swap(a, b);
4         if (dep[a] != dep[b]){
5             int dif = dep[b] - dep[a];
6             for (int i = 0; i < 20; i++){
7                 if (dif & 1) b = p[b][i];
8                 dif >>= 1;
9             }
10        }
11        if (a == b) return a;
12        for (int i = 19; i >= 0; i--){
13            if (p[a][i] != p[b][i]){
14                a = p[a][i];
15                b = p[b][i];
16            }
17        }
18        return p[a][0];
19    }
```

§11. 線段樹 Segment Tree

受限文檔大小，圖例請參閱網站：https://yuihuang.com/segment-tree/

- 【用途】多次查詢區間 [L, R] 的某些資訊，如「區間和」或「區間最大值」。 亦可同時紀錄多種資訊。 資料可以修改 (單點修改或區間修改)。
- 【條件】要查詢的資訊需滿足「結合律」。
- 【實作】
  - 二元樹資料結構 (根節點編號為 1)。
  - 每個節點存一段連續區間 [L, R] 的資訊(e.g. 區間和)，如節點 [1, 2] 存放 A1 + A2。

- 根節點存放區間 [1, n] 的資訊。（下圖假設有 n = 9 筆資料）
- 節點編號：每個點的左兒子的節點編號為 x*2 (x<<1) ，右兒子則為 x * 2 + 1 (x<<1|1) 。
- 陣列 total[n << 2]：存放每段區間的數字總和（開 4 倍陣列避免溢位。）
- build( )：建立線段樹，左子樹紀錄區間 [L, mid]，右子樹紀錄區間 [mid+1, R]。
- pull( )：合併左節點與右節點的值。
- query( )：查詢對應區間 [ql, qr] 的值。
- update( )：更新單點 (位置 pos) 的值 (val)，所有重疊的區間都要一併更新。

- 【練習】HDU 1166. 敌兵布阵【題解】

```
//total：線段樹的每個節點，用來紀錄區間和
//開 4 倍陣列避免溢位
int total[50005 << 2];
```

```
void pull(int x){
    //合併左節點與右節點
    total[x] = total[x<<1] + total[x<<1|1];
}
```

```
 void build(int x, int l, int r){
     if (l == r){
         cin >> total[x];
         return;
     }
     int mid = (l+r) >> 1;
     build(x<<1, l, mid);
     build(x<<1|1, mid+1, r);
     pull(x);
```

```
10      }
```

```
1       int query(int x, int l, int r, int ql, int qr){
2           //[ql, qr]: 查詢區間
3           if (ql <= l && qr >= r){
4               return total[x];
5           }
6           int ret = 0;
7           int mid = (l+r)>>1;
8           if (ql <= mid){
9               ret += query(x<<1, l, mid, ql, qr);
10          }
11          if (qr > mid){
12              ret += query(x<<1|1, mid+1, r, ql, qr);
13          }
14          return ret;
15      }
```

```
1       void update(int x, int l, int r, int pos, int val){
2           if (l == r){
3               total[x] += val;
4               return;
5           }
6           int mid = (l+r) >> 1;
7           if (pos <= mid) update(x<<1, l, mid, pos, val);
8           else update(x<<1|1, mid+1, r, pos, val);
9           pull(x);
10      }
```

【更多練習】HDU 1754. I Hate It【題解】
【更多練習】HDU 3308. LCIS【題解】
【更多練習】HDU 4027. Can you answer these queries?【題解】

§12. 樹狀數組 Binary Indexed Tree

受限文檔大小，圖例請參閱網站：https://yuihuang.com/binary-indexed-tree/

- 【用途】計算動態「前綴和」，比線段樹更節省記憶體，也更容易實作。
- 【概念】給定一個陣列 A[ ] = {A1, A2, A3, A4, A5, A6, A7, A8, A9}
  - 計算區間和 sum({A3, A4, A5, A6}) = sum({ A1, A2, A3, A4, A5, A6}) – sum({A1, A2})
  - 只要對任意的 i，都能算出前綴和 sum({A1, …, Ai})，就可以求得任意區間和。
- 【複雜度】
  - 時間複雜度：O(log(N))
  - 空間複雜度：O(N)
- 【實作】**update**(x, d)：更新 BIT 的值 (第 x 項的值加上 d, delta)，下圖以 A13 + d 為例。
  - 關鍵計算：x += x & (-x)
  - BIT 數組中的 B[13], B[14], B[16] 都包含 A13，因此必須一併更新。

```
1   void update(int x, int d){
2       // update prefix sum in BIT
3       while(x <= N){
4           b[x] += d;
5           x += x & (-x);
6       }
7   }
```

- 【實作】**query**(x)：利用 BIT 計算前 x 項的和，下圖以 sum({A1, A2, …, A13})為例。
- 關鍵計算：x -= x & (-x)
- 答案為 BIT 數組中的 B[13] + B[12] + B[8]。

```
1   int query(int x){
2       // 以 BIT 計算和
3       int ret = 0;
```

```
4         while (x){
5             ret += b[x];
6             x -= x & (-x);
7         }
8         return ret;
9     }
```

【範例】ZeroJudge d794: 世界排名【題解】

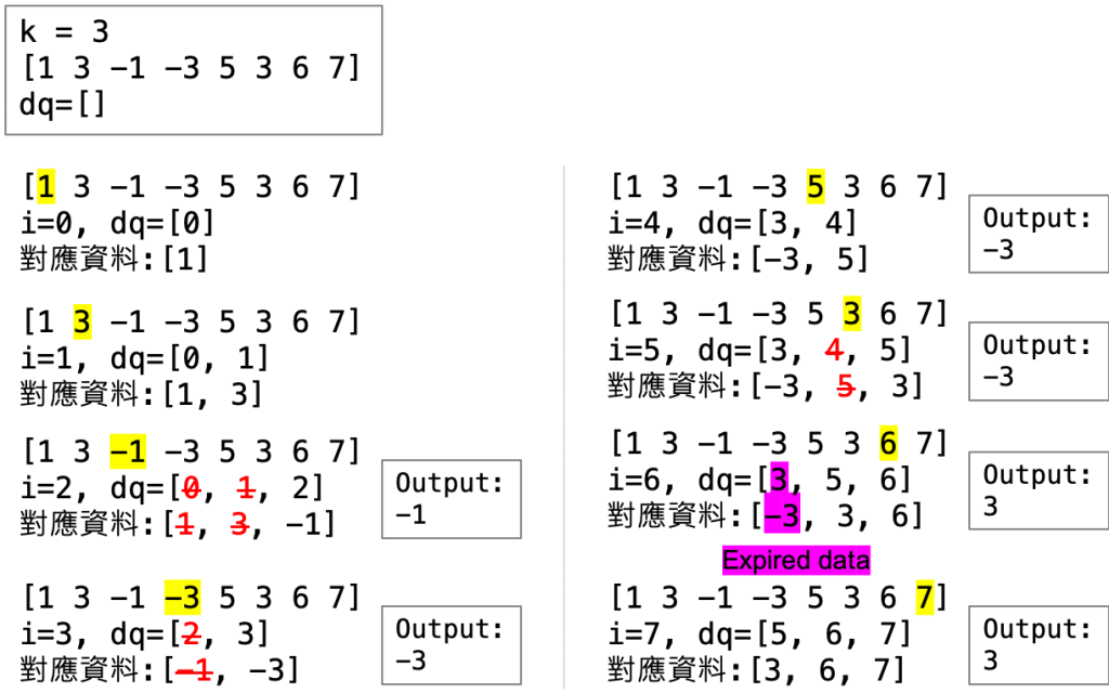
【解題想法】數值離散化 + BIT

```
1     #include <iostream>
2     #include <algorithm>
3     #include <vector>
4     using namespace std;
5     #define ll long long
6
7     int N, M;
8     int b[100001]; //BIT
9     vector<ll> v, d;
10
11    int getID(ll x){
12        //離散化
13        return lower_bound(d.begin(), d.end(), x) - d.begin() + 1;
14    }
15
16    int query(int x){
17        // query prefix sum in BIT
18        int ret = 0;
19        while (x){
20            ret += b[x];
21            x -= x & (-x);
22        }
23        return ret;
24    }
25
26    void update(int x, int d){
27        // update prefix sum in BIT
28        while(x <= N){
29            b[x] += d;
```

```
30              x += x & (-x);
31          }
32      }
33
34      int main(){
35          while (cin >> N){
36              for (int i=1; i<=N; i++){
37                  b[i] = 0; //clear BIT
38              }
39
40              v.clear();
41              for (int i=0; i<N; i++){
42                  cin >> M;
43                  v.push_back(M);
44              }
45              d = v;
46              sort(d.begin(), d.end());
47              // 去除重複的數字
48              // d.erase(unique(d.begin(), d.end()), d.end());
49              for (int i=1; i<=N; i++) {
50                  int idx = getID(v[i-1]);
51                  update(idx, 1);
52                  cout << i - query(idx) + 1 << '\n';
53              }
54          }
55          return 0;
56      }
```

§13. Monotonic Queue 單調隊列

- 【用途】Monotonic Stack 單調棧的升級版，使用 **deque** 實作，單調隊列支援 pop_front( )，可以根據資料 enqueue 的時間順序刪除元素。單調隊列可以用來找尋區間的最大或最小值。
- 【時間複雜度】O(N)
- 【範例】ZeroJudge a146: Sliding Window
  - 注意：本題會出現 k > n 的測資，需額外處理(Line-27)。
  - 運用兩次單調隊列，一次找區間最小值(下圖例)，一次找區間最大值。
  - deque 存放資料的 index，用來維護區間資料的單調遞增性質，因此 a[dq.front()]即為區間最值。
  - Line-17：pop 過時(超出區間)的資料。
  - Line-20：pop 違反單調遞增性的資料。




```
1    #include <iostream>
2    #include <cstring>
3    #include <deque>
4    using namespace std;
5    int a[1000005];
6
7    int main() {
```

```
8           ios_base::sync_with_stdio(0);
9           cin.tie(0);
10          int n, k;
11          while (cin >> n >> k) {
12              for (int i=0; i<n; i++){
13                  cin >> a[i];
14              }
15              deque<int> dq;
16              for (int i=0; i<n; i++){
17                  while (dq.size() && dq.front() <= i - k){
18                      dq.pop_front();
19                  }
20                  while (dq.size() && a[dq.back()] > a[i]) {
21                      dq.pop_back();
22                  }
23                  dq.push_back(i);
24                  if (i == k - 1) cout << a[dq.front()];
25                  if (i > k - 1)  cout << ' ' << a[dq.front()];
26              }
27              if (k > n) {
28                  cout << a[dq.front()];
29              }
30              cout << '\n';
31
32              while (dq.size()) dq.pop_back();
33
34              for (int i=0; i<n; i++){
35                  while (dq.size() && dq.front() <= i - k){
36                      dq.pop_front();
37                  }
38                  while (dq.size() && a[dq.back()] < a[i]) {
39                      dq.pop_back();
40                  }
41                  dq.push_back(i);
42                  if (i == k - 1) cout << a[dq.front()];
43                  if (i > k - 1)  cout << ' ' << a[dq.front()];
44              }
45              if (k > n) {
```

```
46                  cout << a[dq.front()];
47              }
48              cout << '\n';
49          }
50          return 0;
51      }
```

【更多練習】Codeforces 1237D Balanced Playlist【題解】

【更多練習】Codeforces 1195E. OpenStreetMap【題解】

§14. 字典樹 Trie

受限文檔大小，圖例請參閱網站：https://yuihuang.com/trie/

- 【用途】：給定多個字串，接著多次詢問某字串有沒有出現過。
- 【變化】**01 字典樹**：每一層代表一個 bit 或 true/false，用於查找是否出現過。
- 【練習】HDU 4852 Xor Sum【題解】
- 【練習】Codeforces I. Interactive Casino
- 【練習】Google Code Jam 2019 Round 1A C Alien Rhyme【題解】
- 關於字串的一些名詞：字串 string s;
  - 字串長度：s.size(); 或 strlen(s);
  - 子字串 (substring)：字串 s 中一段連續的字串。
  - 子序列 (subsequence)：字串 s 中一段可不連續的字串，但出現順序必須和原字串相同。
  - 前綴(prefix)：從頭開始的一段 substring。
  - 後綴(suffix)：到結尾的一段 substring。

【字典樹實作】

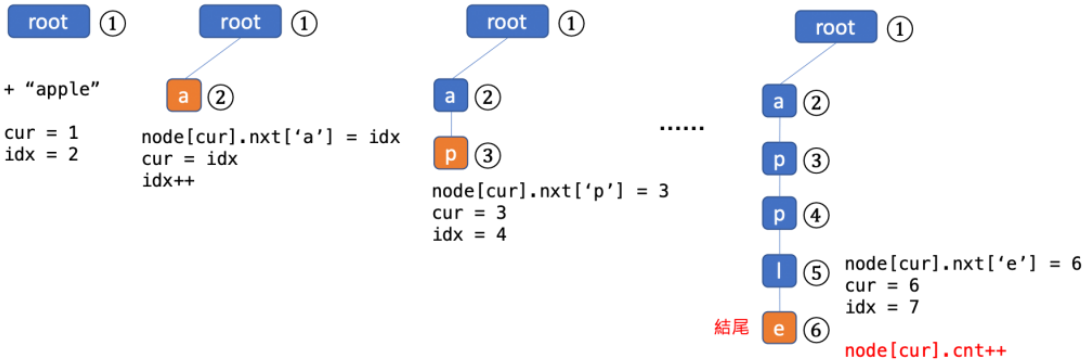
```
struct TrieNode {
    int nxt[26]; //遇到 a~z 時的節點編號(idx)
    int cnt; //以此結尾的字串個數
} node[1000005];

int idx = 2; //root = 1
```

【加入一個字串】

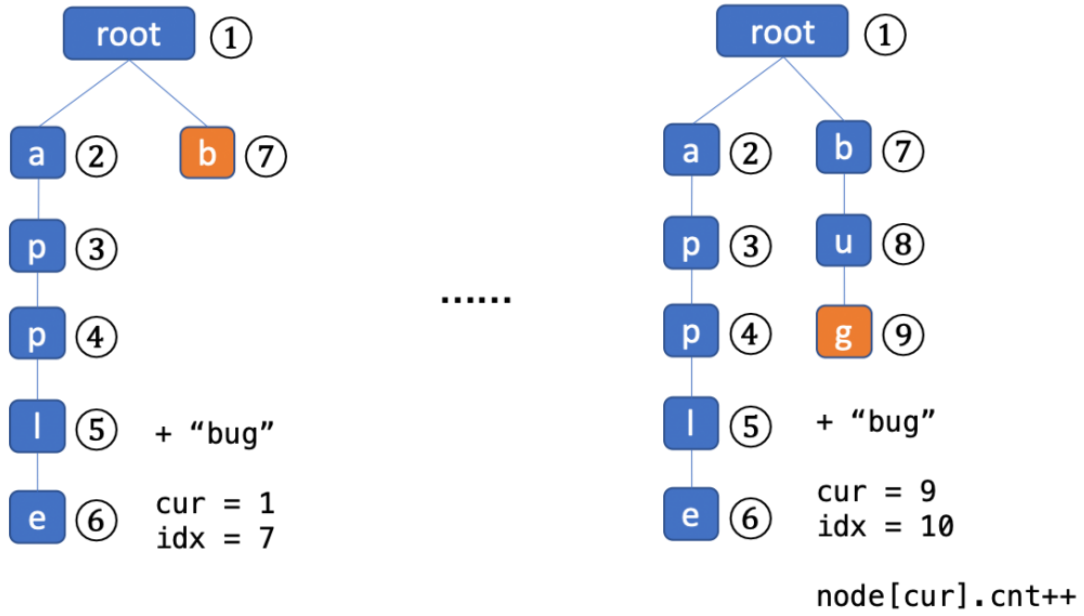
```
voide insert(string s){
    int cur = 1; //從 root 開始
    for(auto i:s) {
        if(node[cur].nxt[i-'a'] == 0){
            node[cur].nxt[i-'a'] = idx; \\開一個新的 node
```

```
6                   cur = idx;
7                   idx++;
8               }
9               else {
10                  cur = node[cur].nxt[i-'a'];
11              }
12          }
13          node[cur].cnt++;
14      }
```

Insert “apple”



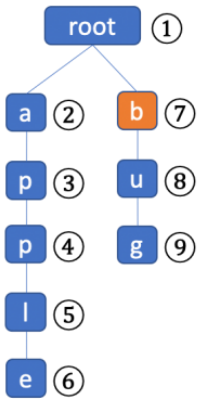
Insert “ape”



【查詢一個字串】

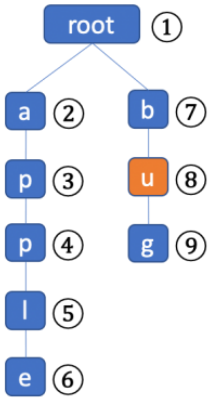
```
1     int query(string s){
2         int cur = 1;
3         for(auto i:s) {
4             if(node[cur].nxt[i-'a'] == 0){
5                 return 0;
6             }
7             else {
8                 cur = node[cur].nxt[i-'a'];
9             }
10        }
11        return node[cur].cnt;
12    }
```

Query "bug"



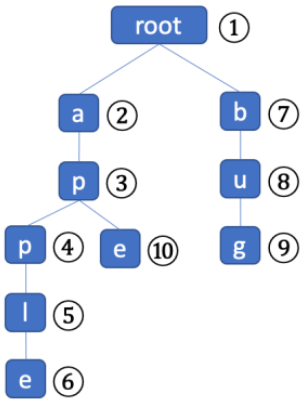
query "bug"
cur = 1
node[cur].nxt['b'] != 0
cur = node[cur].nxt['b']

cur = 7



query "bug"
cur = 7
node[cur].nxt['u'] != 0
cur = node[cur].nxt['u']

cur = 8



query "bug"
cur = 8
node[cur].nxt['g'] != 0
cur = node[cur].nxt['g']

cur = 9
return node[cur].cnt

§15. KMP (Knuth – Morris – Pratt algorithm)

受限文檔大小，圖例請參閱網站：https://yuihuang.com/kmp/

- 【用途】字串匹配
  - 給兩個字串 s，t，計算 t 在 s 的 substring 中**出現的位置(次數)**
- 【做法 1】枚舉起點，暴力比對，複雜度 O(s.size() * t.size()) (TLE)
- 【做法 2】KMP (Knuth–Morris–Pratt algorithm)
  - 先對字串 t 做預處理，減少許多冗余的字元比對
  - 複雜度 O(s.size() + t.size())
- 【預處理】**Failure function** 共同前後綴
  - failure function：若比到此位置才失敗，下一個可以繼續嘗試的位置
  - 【例】a a b a a c a a b a a c d
  - 【例】-1 0 -1 0 1 -1 0 1 2 3 4 5 -1 以此位置結尾的 substring 對整個字串的前綴最長可匹配到哪個位置
- 比對失敗時，對該位置的 failure function 繼續嘗試，複雜度為：
  - 比對成功，位置向後推一格
  - 比對失敗，可能需要查多次 failure 直至推至 -1
  - 因為位置一次只能向後推一格，失敗的次數最多和字串長度相同
- 【練習】HDU 1867 A + B for you again
- 【練習】HDU 2203 亲和串【題解】
- 【練習】HDU 3336 Count the string【題解】

【實作】 build Failure function

```
vector<int> build(string &t) {
    vector<int> F(t.size());
    F[0] = -1;

    for (int i = 1, pos = -1 ; i < t.size() ; i++) {
        while (~pos && t[i] != t[pos + 1])
            pos = F[pos];

        if (t[i] == t[pos + 1])
            pos++;

        F[i] = pos;
    }
    return F;
}
```

【實作】 match

```
bool match(string &s, string &t, vector<int> &F) {
    for (int i = 0, pos = -1 ; i < s.size() ; i++) {
        while (~pos && s[i] != t[pos + 1])
            pos = F[pos];

        if (s[i] == t[pos + 1])
            pos++;

        if (pos + 1 == (int)t.size())
            return true;
    }
    return false;
}
```

§16. 二分圖的最大匹配

受限文檔大小，圖例請參閱網站：https://yuihuang.com/bipartite-graph-maximum-matching/

- 【範例】TIOJ 1089 . Asteroids
- 用 DFS 找二分圖的最大匹配數目
  - g[xi][yj]：1 表示 xi 和 yj 之間可連通，反之為 0。
  - a[yj] = xi：表示把 yj 配對給 xi。
  - vis[yj]：表示 yj 已經被拜訪過。**檢查每個新的 x 前都要先初始化 vis[ ]** (Line-27)。
- 枚舉 x
  - dfs (x)：枚舉 y，進行匹配。
  - (Line-11) 如果找到一個 y (= i) 是未曾與別的 x 配對 (a[i] = 0)，則可以把 x 配對給 i (a[i] = x)。
  - (Line-11) 如果 y (= i) 曾經與別的 x 配對 (a[i] != 0)，則試試看能不能改變該 x 的配對方式，dfs (a[i])。成功改配的話，還是可以把 x 配對給 i (a[i] = x)。
- 【圖例】
  - dfs (x1)：a[y2] = x1, vis[y2] = 1
  - dfs (x2)：a[y3] = x2, vis[y3] = 1
  - dfs (x3)：a[y1] = x3, vis[y1] = 1
  - dfs (x4)：a[y1] != 0 –> dfs (a[y1] = x3) –> (此時 vis[1] = vis[2] = vis[3] = 1) a[y4] = x1, a[y1] = x4

```
1    #include <iostream>
2    #include <cstring>
3    using namespace std;
4
5    int n, k, r, c, g[505][505], a[505], vis[505], ans;
6
7    bool dfs(int x){
8        for (int i = 1; i <= n; i++){
```

```
9               if (g[x][i] == 1 && vis[i]==0){
10                  vis[i] = 1;
11                  if (a[i]==0 || dfs(a[i])){
12                      a[i] = x;
13                      return 1;
14                  }
15              }
16          }
17          return 0;
18      }
19
20      int main() {
21          cin >> n >> k;
22          for (int i = 0; i < k; i++){
23              cin >> r >> c;
24              g[r] = 1;
25          }
26          for (int i = 1; i <= n; i++){
27              memset(vis, 0, sizeof(vis));
28              if (dfs(i)) ans++;
29          }
30          cout << ans << "\n";
31      }
```