HỘI THẢO CÁC TRƯỜNG CHUYÊN VÙNG DUYÊN HẢI

CHUYÊN ĐỀ MÔN TIN HỌC

TRƯỜNG THPT CHUYÊN HÙNG VƯƠNG – PHÚ THỌ

QUY HOẠCH ĐỘNG BITMASK

Việt Trì, tháng 9 năm 2020

MỤC LỤC

CHUYÊN ĐỀ: QUY HOẠCH ĐỘNG BITMASK

I. TÓM TẮT

1.1. Khái niệm cơ bản của bitmask

- Bitmask hay còn được gọi là bộ Boolean nhỏ, nhẹ (hỗ trợ riêng cho C/ C++/ Java) là một dãy bit thể hiện nhiều trạng thái của một đối tượng.
- Như chúng ta đã biết, một số nguyên trong máy tính được biểu diễn bằng một chuỗi hoặc nhiều chuỗi bit với nhau. Do đó, chúng ta có thể sử dụng các số nguyên để biểu diễn cho bitmask. Sau đó, tất cả các trạng thái trong một bitmask chỉ liên quan đến các phép thao tác bit. Điều này làm cho nó trở thành một lựa chọn hiệu quả hơn so với các cấu trúc dữ liệu: vector, set, bitset, … trong C++. Tốc độ của bitmask có thể nhanh hơn so với các CTDL trên làm cho nó trở nên rất quan trọng trong lập trình cạnh tranh.
- Chúng ta đã biết rằng một số nguyên được biểu diễn bởi chuỗi bit. Bit thứ 1 sẽ cho ta biết trạng thái của phần tử thứ 1 có được chọn hay không, tương tự với các phần tử thứ 2, thứ 3, v.v. Ví dụ, xét một tập hợp A = {1, 2, 3, 4, 5} có 5 phần tử và chúng ta chọn ra tập con B = {1, 3, 4}. Bitmask biểu diễn điều này dưới dạng dãy nhị phân là 01101 hoặc 13 ở dạng thập phân).
- Lưu ý: Trong chuyên đề này, các dãy bit được tính từ bên phải (vị trí nhỏ nhất) rồi tiến dần sang trái. Ví dụ là 0001 thì sẽ là 1 ở dạng thập phân.

1.2. Phép toán thao tác bit

- Nếu trong các hệ thập phân có phép toán cộng, trừ, nhân, chia ,… thì trong hệ nhị phân chúng ta có phép toán and, or, not, xor, dịch trái, dịch phải và được biểu diễn cụ thể hơn ở dưới đây.

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

Phép AND:

+ Kí hiệu là: &

A	B	A & B
0	0	0
0	1	1
1	0	1
1	1	1

Phép OR:

+ Kí hiệu là: |

+ Ví dụ:

A: 0101 (số thập phân 5)

B: 0011 (số thập phân 3)

A & B: 0001 (số thập phân 1)

+Ví dụ:

A: 0101 (số thập phân 5)

B: 0011 (số thập phân 3)

A | B: 0111 (số thập phân 7)

A	B	A ~ B
0	0	0
0	1	0
1	0	0
1	1	1

Phép NOT:

+ Kí hiệu là: ~

+ Ví dụ:

A: 0011 (số thập phân 3).

~A: 1100 (số thập phân 12).

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

Phép XOR:

+ Kí hiệu là: ^

+ Ví dụ:

A: 0010 (số thập phân 2)

B: 1010 (số thập phân 10)

A ^ B: 1000 (số thập phân 8)

- Phép dịch trái <<: + Kí hiệu: << + Phép dịch trái n bit tương đương với phép nhân cho 2^n. + Ví dụ: A: 0000 1100 (số tp 12) ← B = A << 2: 0011 0000 (số tp 48).	- Phép dịch phải >>: + Kí hiệu: >> + Phép dịch trái n bit tương đương với phép chia cho 2^n. + Ví dụ: A: 0000 1100 (số tp 12). → B = A >> 2: 0000 0011 (sô tp 3).

1.3. Quy hoạch động bitmask

- Kĩ thuật dùng Bitmask là một trong những kĩ thuật quan trọng mà chúng ta cần phải biết.
- Chúng thường được nhận dạng khi với các thông số cho trước trong đề ta có thể giải quyết bài toán bằng độ phức tạp cấp số nhân.
- Trong kĩ thuật này, ta cần lưu ý rằng các bài toán con sẽ được lưu trong một mảng được gọi là mảng trạng thái. Trạng thái hiện tại sẽ cập nhật dữ liệu của trạng thái trước. Ví dụ: 000010 là trạng thái trước của 100010.
- Kĩ thuật này có thể giải thay thế một số bài toán quay lui, nhánh cận và giải nhiều bài toán đếm với cấu hình nhỏ.
- Phân loại: 2 loai chính:
    + Loại thứ 1: Là loại mà cấu hình vị trí của giá trị không thay đổi - tức là cách chọn cố định.
    + Loại thứ 2: Là loại mà phụ thuộc vào cách sắp xếp thứ tự giá trị để bài toán tối ưu.
- Thông thường, bảng sẽ được xây dựng như sau:
    + F[X]: chỉ lưu giá trị tại trạng thái X, chủ yếu dùng cho loại 1.
    + F[X, I]: lưu giá trị tại thạng thái X là I được thêm vào cuối cùng.
- Ví dụ: Xét bài toán: Cho N người và N công việc, mỗi nhiệm vụ được phân công cho từng người. Nói cách khác, chúng ta có một ma trận cost với kích thước N ×N, với cost[i][j] được hiểu là chi phí để người i hoàn thành nhiệm vụ j. Bây giờ chúng ta cần sắp xếp mỗi nhiệm vụ cho mỗi người sao cho tổng

giá trị là nhỏ nhất. Lưu ý rằng mỗi nhiệm vụ sẽ được giao cho một người và mỗi người chỉ được chọn một nhiệm vụ.

+ Ngay khi đọc xong đề, chúng ta sẽ nghĩ ngay đến thuật toán duyệt trâu là xét tất cả các hoán vị của nó. Thuật toán được đưa ra ở dưới đây:

```
assign(N,  cost)
   for i = 0 -> N
      assignment[i] = i          //giao nhiệm vụ i cho người thứ i
   res = INFINITY
   for j = 0 -> factorial(N)     // duyệt tất cả các hoán vị
      total_cost = 0
      for i = 0 to N
         total_cost = total_cost + cost[i][assignment[i]]
      res = min(res, total_cost)
      generate_next_greater_permutation(assignment)  // đi tới hoán vị tiếp theo
   return res
```

+ Độ phức tạp của thuật toán trên là O(N!) là tệ nhất.
+ Bây giờ chúng ta sẽ sử dụng QHĐ bitmask. Ta gọi dp(k, mask) là tổng giá trị nhỏ nhất khi giao việc xong cho những người từ 0 đến k - 1, và mask là một số nhị phân với mỗi bit thể hiện trạng thái của nhiệm vụ thứ i (đã được chọn hay chưa). Nếu bit thứ i bật thì nhiệm vụ thứ i đã được chọn, ngược lại thì nhiệm vụ thứ i đã được chọn.
+ Với mỗi trạng thái dp(k, mask), ta sẽ cần tìm nhiệm vụ cho người thứ i, và đương nhiên, ta chỉ được chọn những nhiệm vụ j mà chưa được chọn, hay là bit thứ j chưa bật. Do đó, ta có công thức quy hoạch động:

```
dp(k + 1, mask | (1 << i)) = min(dp(k + 1, mask | (1 << i)), dp(k, mask) + cost[k][i]).
```

+ Một điều cần lưu ý là k lại đúng bằng số bit bật của mask, do đó nó được rút gọn thành dp(mask). Do đó, ta lại có công thức sau:

```
dp(mask | (1 << i)) = min(dp(mask | (1 << i)), dp(mask) + cost[x][i])
(với x là số bit được bật trong mask.)
```

+ Thuật toán được mô tả kĩ hơn trong mã giả dưới đây:

```
assign(N, cost)
  for i = 0 -> power(2,N)
    dp[i] = INFINITY
  dp[0] = 0
  for mask = 0 -> power(2, N)
    x = count_set_bits(mask) // đếm số bit bật trong mask.
    for j = 0 -> N
      if jth bit is not set in i // nếu bit thứ j chưa được bật
        dp[mask|(1<<j)] = min(dp[mask|(1<<j)], p[mask]+cost[x][j])
  return dp[power(2,N)-1] // kết quả của bài toán.
```

+ Độ phức tạp của thuật toán này là $O(2^n \times n)$ và bộ nhớ là $O(2^n)$, nhanh hơn so với thuật toán duyệt tất cả các cấu hình.

II. CÁC BÀI TẬP MINH HỌA

2.1. Bài 1: LEM3

1.1. Đề bài:

Trong kì nghỉ hè năm nay Sherry được bố thưởng cho một tour du lịch quanh N đất nước tươi đẹp với nhiều thắng cảnh nổi tiếng (vì Sherry rất ngoan). Tất nhiên Sherry sẽ đi bằng máy bay.

Giá vé máy bay từ đất nước i đến đất nước j là Cij (dĩ nhiên Cij có thể khác Cji). Tuy được bố thưởng cho nhiều tiền để đi du lịch nhưng Sherry cũng muốn tìm cho

mình một hành trình với chi phí rẻ nhất có thể để dành tiền mua quà về tặng mọi người (các chuyến bay của Sherry đều được đảm bảo an toàn tuyệt đối).

Bạn hãy giúp Sherry tìm một hành trình đi qua tất cả các nước, mỗi nước đúng một lần sao cho chi phí là bé nhất nhé.

Input:

- Dòng đầu tiên chứa hai số nguyên N ($5 < N < 16$).
- Dòng thứ *i* trong *N* dòng tiếp theo: Gồm *N* số nguyên, số thứ *j* là C_{ij} ($0 < C_{ij} < 10001$)

Output:

- In ra một số nguyên –chi phí bé nhất tìm được.

Examples:

LEM3.inp	LEM3.out
6 0 1 2 1 3 4 5 0 3 2 3 4 4 1 0 2 1 2 4 2 5 0 4 3 2 5 3 5 0 2 5 4 3 3 1 0	8

1.2. Thuật toán hiệu quả nhất

Đây là một bài QHĐ bitmask cơ bản. Ta gọi dp(u, mask) là đường đi bé nhất mà đã đi qua các đỉnh đã bật trong mask và đỉnh cuối cùng đi qua là u. Từ đó, ta có công thức sau:

dp(v, mask | (1 << v)) = min(dp(v, mask | (1 << v)), dp(u, mask) + d[u][v])
(với d[u][v] là đường đi từ u đến v).

Độ phức tạp của bài toán này là $O(n^2 \times 2^n)$.

1.3. Code

```
#include <bits/stdc++.h>

using namespace std;
#define endl '\n'
#define task "LEM3"
#define MASK(i) (1LL << (i))
#define c_bit(i) __builtin_popcountll(i) // đếm số bit đang bật
```

```
#define BIT(x, i) ((x) & MASK(i)) // trạng thái của bit thứ i trong x
#define SET_ON(x, i) ((x) | MASK(i)) // bật bit thứ i trong x
#define SET_OFF(x, i) ((x) & ~MASK(i)) //  tắt bit thứ i trong x
const int MAXN = 16;
const int INF = 1e9 + 7;
int N;
int C[MAXN][MAXN];
void Input()
{
	cin >> N;
	for(int i = 0; i < N; i++)
		for(int j = 0; j < N; j++)
			cin >> C[i][j];
}
int f[MAXN][MASK(MAXN)];
void Output()
{
	for(int i = 0; i < N; i++)
		for(int j = 0; j < MASK(N); j++)
			f[i][j] = INF;
	for(int i = 0; i < N; i++)
		f[i][MASK(i)] = 0;
	for(int mask = 0; mask < MASK(N); mask++)
		for(int u = 0; u < N; u++)
			for(int v = 0; v < N; v++)
				if(!BIT(mask, v))
					f[v][SET_ON(mask,v)]                    =min(f[v]
[SET_ON(mask, v)], f[u][mask] + C[u][v]);
	int ans = INF;
	for(int i = 0; i < N; i++)
		ans = min(ans, f[i][MASK(N) - 1]);
	cout << ans << endl;
}
int main()
{
   ios_base::sync_with_stdio(false);
   cin.tie(0);
   freopen(task".inp", "r", stdin);
   freopen(task".out", "w", stdout);
   Input();
   Output();
   return 0;
```

```
}
```

1.4. Test trong thư mục đính kèm

2.2. Bài 2: KEYBOARD

2.1 Đề bài:

Bạn có một mật khẩu mà bạn hay phải gõ − một xâu s có độ dài là n. Mỗi kí tự của xâu này là một trong m kí tự đầu tiên trong bản chữ cái Latin in thường. Vì bạn phải gõ mật khẩu rất nhiều, bạn muốn mua một bàn phím mới. Một bàn phím là một hoán vị của m chữ cái Latin đầu tiên. Ví dụ, nếu $m = 3$ thì có 6 bàn phím khác nhau là: abc, acb, bac, bca, cab và cba. Vì bạn chỉ gõ mật khẩu với một ngón tay, bạn cần thời gian để di chuyển từ một kí tự trong mật khẩu sang kí tự tiếp nối nó. Thời gian bạn dành ra để di chuyển từ kí tự s_i sang kí tự s_{i+1} bằng với khoảng cách của hai kí tự này trên bàn phím. Thời gian tổng cộng mà bạn cần để gõ mật khẩu này ra với một bàn phím nhất định thì được gọi là độ chậm của bàn phím đó.

Định nghĩa cụ thể hơn, độ chậm của một bàn phím thì bằng

$$\sum_{i=2}^{n} ¿\, pos(s_{i-1}) - pos(s_i) \vee ¿¿$$

Với $pos(x)$ là vị trí của kí tự x trên bàn phím.

Ví dụ, nếu s là aacabc và bàn phím là bac, thì tổng thời gian để gõ mật khẩu là:

$|pos(a) - pos(a)| + |pos(a) - pos(c)| + |pos(c) - pos(a)| +$
$|pos(a) - pos(b)| + |pos(b) - pos(c)|$
$= |2 - 2| + |2 - 3| + |3 - 2| + |2 - 1| + |1 - 3| = 0 + 1 + 1 + 1 + 2 = 5$

Trước khi mua một bàn phím mới, bạn muốn biết độ chậm thấp nhất của một bàn phím có thể là bao nhiêu.

Input:

- Dòng đầu tiên chứa hai số nguyên n và m ($1 \le n \le 10^5$; $1 \le m \le 20$).
- Dòng thứ hai chứa một xâu s có n kí tự. Mỗi kí tự là một trong m chữ cái Latin đầu tiên (in thường).

Output:

- In ra một số nguyên – độ trễ bé nhất mà một bàn phím có thể có.

Examples:

KEYBOARD.inp	KEYBOARD.out
6 3 aacabc	5

6 4 aaaaaa	0
15 4 abacabadabacaba	16

Note:

- Ví dụ đầu tiên được xem xét trong đề bài.
- Trong ví dụ thứ hai, mọi bàn phím đều có độ chậm là 0.
- Trong ví dụ thứ ba, bàn phím tốt nhất là bacd.

2.1. Thuật toán hiệu quả nhất

- Nhận xét là xâu s thực sự không quá quan trọng, chỉ cần biết được $c[x][y]$ là số lần ta đang từ kí tự x trong xâu s mà chuyển sang kí tự tiếp theo là y. Kết quả cho một bàn phím nhất định sẽ là:

$$sum(abs(pos[x] - pos[y]) \times (c[x][y] + c[y][x])).$$

- Bây giờ nhận thấy là nếu ta biết được x sẽ đứng trước hay sau y trong bàn phím thì ta có thể tách cái tổng ra làm hai phần chỉ liên quan đến $pos[x]$ và $pos[y]$, từ đó có thể tính được các giá trị đóng góp vào độ trễ của $pos[x]$ và $pos[y]$ mà không quan tâm đến giá trị còn lại.
- Cụ thể, ta sẽ đi xây dựng bàn phím từ trái sang phải. Nếu ta hiện tại có xâu bàn phím là a và tập các số chưa được thêm vào là S thì có thể mô tả các cách chọn như sau:
  + Với mỗi $x \in S$, nếu ta thêm x vào xâu a thì chi phí mà x phải trả là:

$$\sum_{y \in a} pos[x] \times (c[x][y] + c[y][x])$$

$$+¿$$

$$\sum_{y \in ¿¿} ¿$$

(Vì x xuất hiện sau mỗi kí tự trong a nên cộng vị trí của x cho mỗi lần di chuyển giữa một kí tự trong a đến x và vì x xuất hiện trước các kí tự trong $(S \backslash \{x\})$ nên phải trừ đi vị trí của x.)

- Điều này cũng có nghĩa rằng xâu a là gì không quan trọng, chỉ cần biết tập hợp các chữ cái có trong a.
- Bài toán bây giờ trở thành từ tập rỗng, thêm dần các kí tự, mỗi lần thêm thì sẽ phải trả chi phí nhất định tùy vào việc đang thêm vào tập nào, hỏi cách thêm thế nào mà

tốn ít chi phí nhất. Thì đây là một bài quy hoạch động được, cái chính là lưu các tập như thế nào.

- Việc lưu tập là cái chỗ mà bitmask có tác dụng. Cụ thể, ta lưu lại một dãy bit sao cho bit 0 thì phần tử đó không có trong tập, bit 1 thì phần tử đó có trong tập. Thao tác thêm từ một tập thì đơn giản là bật một cái bit lên để chuyển đến tập mới. Rõ ràng thì mỗi bitmask sẽ ứng với một số, vì thế ta có thể lưu trữ các tập bằng một mảng số nguyên mà không tốn kém nhiều chi phí để đọc / ghi vào các tập.
- Biết cách lưu tập hợp và biết cách chuyển trạng thái thì ta có thể thực hiện quy hoạch động một cách đơn giản.

2.3. Code

```
#include <bits/stdc++.h>
// #define int long long
#define double long double
#define ii pair <int, int>
#define zz pair <int, ii>
#define X first
#define Y second
#define a(i, j) aa[((i) - 1) * (N) + (j)]
#define BIT(mask, i) ((mask) & (1ll << (i)))
#define ONBIT(mask, i) ((mask) | (1ll << (i)))
#define OFFBIT(mask, i) ((mask) &~ (1ll << (i)))
#define Task "KEYBOARD"
using namespace std;
const int oo = 1e9 + 7;
const int eps = 1e-9;
string S;
int N, M;
int save[21][(1 << 20) + 1], CNT[(1 << 20) + 1];
int MinBit(int mask) {
    int mb = mask & -mask;
    for (int i = 0; i < M; ++i)
        if (BIT(mask, i))
            return i;
}
void Cal() {
    for (int mask = 1; mask < (1 << M); ++mask) {
        int cnt1 = 0, cnt2 = 0;
        int oldmask = mask - (mask & -mask);
```

```
        int t = MinBit(mask);
        for (int i = 0; i < M; ++i) {
            if (BIT(mask, i))
                continue;
            int t1 = t, t2 = i;
            if (t1 > t2)
                swap(t1, t2);
            cnt1 += save[t1][t2];
        }
        for (int i = 0; i < M; ++i) {
            if (!BIT(mask, i))
                continue;
            int t1 = t, t2 = i;
            if (t1 > t2)
                swap(t1, t2);
            cnt2 += save[t1][t2];
        }
        CNT[mask] = CNT[oldmask] + cnt1 - cnt2;
    }
}
int solve(int x, int mask) {
    if (x > M)
        return 0;
    if (save[x][mask] != -1)
        return save[x][mask];
    int cur = oo;
    for (int i = 0; i < M; ++i) {
        if (BIT(mask, i))
            continue;
        int newmask = ONBIT(mask, i);
        cur = min(cur, solve(x + 1, newmask) + CNT[newmask]);
    }
    return save[x][mask] = cur;
}
main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    freopen(Task".inp", "r", stdin);
    freopen(Task".out", "w", stdout);
    cin >> N >> M;
    cin >> S;
    for (int i = 1; i < S.size(); ++i) {
```

```
        if (S[i] == S[i - 1])
            continue;
        int t1 = (int)(S[i - 1] - 'a');
        int t2 = (int)(S[i] - 'a');
        if (t1 > t2)
            swap(t1, t2);
        ++save[t1][t2];
    }
    Cal();
    memset(save, -1, sizeof(save));
    cout << solve(1, 0);
}
```

2.4. Test trong thư mục đính kèm

2.3. Bài 3: BMAGIC

3.1.Đề bài:

(https://codeforces.com/problemset/problem/453/B)

Cô công chúa đã đi đến Celestia và Lâu đài của Luna để tìm kiếm chiếc rương kho báu của chú kì lân.

Dãy số nguyên dương b_i là hài hòa khi và chỉ khi với mỗi hai phần tử của dãy có ước chung lớn nhất của chúng bằng 1. Theo câu truyện thần thoại có thật ngày xưa, mật khẩu của chiếc rương là dãy số hòa hòa b_i mà sao cho biểu thức sau được nhỏ nhất:

$$\sum_{i=1}^{n} |a_i - b_i|.$$

Bạn được cung cấp 1 dãy a_i hãy giúp công chúa tìm mật khẩu thích hợp.

Input :

- Dòng đầu tiên chứa số nguyên n. ($1 \le n \le 100$)
- Dòng tiếp theo chứa n số nguyên ai ($1 \le a_i \le 30$)

Output :

- Mã khóa là dãy số b sao cho tổng trên là bé nhất. Nếu có nhiều câu trả lời in ra bất kì

Examples:

BMAGIC.inp	**BMAGIC.out**
5 1 6 4 2 8	1 5 3 1 8

3.2. Thuật toán hiệu quả nhất

- Bài này ta sử dụng QHĐ bitsmask.
- Ta thấy số trong dãy b phù hợp làm mật khẩu lớn nhất tối đa là 60 (2 * ai)
- Có 17 số nguyên tố từ 1 đến 60 ta có thể xây dựng được công thức quy hoạch động
- Dp[i][mask] mình đã xét đến vị trí i và đã sử dụng được các số nguyên tố vị trí bit 1 bật trong mask

3.3.Code

```
#include <bits/stdc++.h>
#define task "BMAGIC"
#define FOR(i,a,b) for(int i = (a); i <= (b); i ++)
#define MASK(i) (1LL << (i))
#define BIT(x, i) (((x) >> (i)) & 1) // kiểm tra bit thú i của số x
#define SET_ON(x, i) ((x) | MASK(i)) // bật bit thứ i của số x
#define SET_OFF(x, i) ((x) & ~MASK(i)) // tắt bit thứ i của số x

// #define int long long
using namespace std;
const int oo = (int)1e9;
const int mod = 1e9 + 7;

void IOS() {
   ios_base::sync_with_stdio(0);
   cin.tie(0);
   cout.tie(0);
}
void FRE() {
   freopen(task".inp", "r", stdin);
   freopen(task".out", "w", stdout);
}

const int N = 105;
const int MAXX = 60;
int n;
int A[N];
int f[N][MASK(17)];
int prime[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59}; // Khởi
tạo các số nguyên tố <= 60
int pri_mask[MAXX];
int trave[N][MASK(17)]; // mảng truy vế
void minimize(int &a,const int &b) {
        if(a > b) {
                a = b;
        }
}
main()
{
   IOS();
```

```
    FRE();


    cin >> n;
    FOR(i,1,n) {
        cin >> A[i];
    }
    FOR(i,0,MAXX) {
        FOR(j,0,16) {
            if(i % prime[j] == 0) {
                pri_mask[i] |= (MASK(j));  // bật các bit là thứ thự của số nguyên tố tạo ra số i
            }
        }
    }

    FOR(i,0,n) {
        FOR(j,0,MASK(17)-1) {
            f[i][j] = oo; // khởi tạo mảng QHĐ
        }
    }
    f[0][0] = 0;
    FOR(i,1,n) {
        FOR(j,0,MASK(17)-1) {
            FOR(k,1,MAXX) {
                if((pri_mask[k] & j)) continue; // kiểu tra nếu 2 mask có cùng 1 vị trí bật thì không xét
                if(f[i][pri_mask[k] | j] > f[i-1][j] + abs(A[i]-k)) {
                    trave[i][pri_mask[k] | j] = k; // đánh dấu truy về là chọn số k ở vị trí i
                    f[i][pri_mask[k] | j]  = f[i-1][j] + abs(A[i]-k); // chọn số k cập nhập vào mảng QHĐ
                }
            }
        }
    }

    int Ans = oo;
    int Ans_mask;
    FOR(j,0,MASK(17)-1) {
        if(Ans> f[n][j]) {
            Ans = f[n][j]; // lấy kết quả
```

```
                Ans_mask = j; // tìm mask sao cho các số nguyên tố tại vị trí bật bit là
các số khởi tạo ra dãy đáp án
            }
        }
        vector<int> Res;
        // truy vế
        for(int i = n, mask = Ans_mask; i >= 1; i --) {
            Res.push_back(trave[i][mask]);
            mask = mask ^ pri_mask[trave[i][mask]]; // xóa các bit của số vừa lấy
        }

        for(int i = n - 1; i >= 0; i -- ) {
            cout << Res[i] << " ";
        }

        return 0;
}
/// Change Dream Magic Dr_Unicorn
```

3.4.Test ở thư mục gửi kèm

2.4. Bài 4: LFO

4.1. Đề bài

(https://codeforces.com/problemset/problem/8/C)

Lena thích nó khi mọi thứ đều được sắp đạt, và nhìn thấy sự sắp đặt ở mọi nơi. Cô đã được nhận vào một trường đại học và thấy căn phòng của mình rất bừa bộn. Tất cả mọi thứ trong túi của cô nằm rải rác khắp trong phòng . Tật nhiên cô muốn nhặt tất cả chúng vào trong túi của cô. Vấn đề ở đây là cô không thể mang quá 2 vật phẩm cùng một lúc và không thể di chuyển được cái túi của cô. Cô nhờ bạn giúp. Khi bạn lấy 1 vật phẩm, cô cũng không cho phép đặt nó bất cứ đâu ngoại trừ túi của cô.

Bạn đã nhận được tọa đồ của các vật phẩm trong đơn vị tọa độ Cartesian. Nó cho biết rằng thời gian đi giữa 2 vật là bình phương khoảng cách của 2 vật đấy (đồ thứ i và đồ thứ j cách nhau $(x_i-x_j)^2 + (y_i-y_j)^2$. Nó cho biết tọa độ ban đầu của cái túi và cô giống nhau. Bạn hãy tìm ra một cách sắp đặt các hành động sao cho có thể cho tất các các đồ vật vào túi thời gian ngắn nhất.

Input :

- Dòng đầu tiên la tọa độ của cái túi x_s , y_s.
- Dòng thứ hai chưa số n ($1 \le n \le 24$) Số các vật phẩm cô có.
- Tiếp theo là n dòng chứa tọa độ các vật phẩm. Giá trị tuyệt đối của các tọa độ không quá 100 . Tất cả các toạ độ các vật khác nhau và tất cả là số nguyên

Output :

- Dòng đầu tiên chứa một số duy nhất là số thời gian bé nhất cần để cho các vật phẩm vào trong túi
- Trong lần thứ 2 lần lượt là thứ tự các vị trí cô ấy đi để (vị trí cái túi là 0). Nếu có nhiều câu trả lớn in ra cách bất kì

Examples :

LFO.inp	**LFO.out**
1 1 3 4 3 3 4 0 0	32 0 1 2 0 3 0

4.2.Thuật toán hiệu quả nhất

- Ta sẽ QHĐ bitsmask
- Và có 1 nhận xét từ vị trí cái túi đến 2 vị trí hay 1 vị trí bất khì rồi trở về không thì thứ tự đến các vị trí đấy đấy thay đổi cũng không thể thay đổi thời gian đi của các vị trí.
- Đặt Dp[mask] là thời gian ngắn nhất để có thể cho được các đồ vật tại vị trí bật trong mask và trong túi. Với mỗi mask hay lấy thêm 1 vật phẩm chưa được thu nhặt và 1 vật phẩm bất kì.
- Độ phức tạp bài toán là $O(n \times n^2)$.

4.3. Code

```
#include <bits/stdc++.h>
#define task "LFO"
#define FOR(i,a,b) for(int i = (a); i <= (b); i ++)
#define MASK(i) (1LL << (i))
#define BIT(x, i) (((x) >> (i)) & 1) // kiểm tra bit thú i của số x
#define SET_ON(x, i) ((x) | MASK(i)) // bật bit thứ i của số x
#define SET_OFF(x, i) ((x) & ~MASK(i)) // tắt bit thứ i của số x
```

```
#define ii pair<int, int>
#define fi first
#define se second
// #define int long long
using namespace std;
const int oo = (int)1e9;
const int mod = 1e9 + 7;

void IOS() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
}
void FRE() {
    freopen(task".inp", "r", stdin);
    freopen(task".out", "w", stdout);
}

const int N = 26;

int n;
int len[N][N];
ii XY[N];
int Dp[(1 << 24)];
int Tv[(1 << 24)];
main()
{
    IOS();
    FRE();
    cin >> XY[0].fi >> XY[0].se;
        cin >> n;
        FOR(i,1,n) {
                cin >> XY[i].fi >> XY[i].se;
        }
        FOR(i,0,n) {
                FOR(j,0,n) {
                        int h = XY[i].fi - XY[j].fi;
                        int t = XY[i].se - XY[j].se;
                        len[i][j] = h * h + t * t;    // Khởi tạo mảng thời gian đi từ điểm i
đến điểm j
                }
        }
```

```
        memset(Dp, 0x3f, sizeof Dp);
        Dp[0] = 0;
        FOR(mask, 0, MASK(n)-1) {
                FOR(i,0,n-1) {
                        if(BIT(mask,i)) continue;
                        FOR(j,i,n-1) {

                                if(BIT(mask,j)) continue;
                                // chọn ra 2 vị trí ( có thể trùng nhau ) mà chưa được cất vào
túi
                                int nw_mask = MASK(i) | MASK(j) | mask;
                                int  Res  =  len[0][i+1]  +  len[i+1][j+1]  +  len[j+1][0]  +
Dp[mask];
                                if(Dp[nw_mask] > Res) {
                                        // chọn lấy 2 vật đấy về tủi
                                        Dp[nw_mask] = Res;;
                                        // đánh dấu truy vết các đồ vật lấy
                                        Tv[nw_mask] = mask;
                                }
                        }
                        break;
                }
        }
        cout << Dp[MASK(n)-1] << '\n';
        cout << 0 << " ";
        int mask = MASK(n)-1;
        int j = 6;
        // truy vết lại kết quả
        while(mask != 0) {
                int ck_mask = Tv[mask] ^ mask; // tắt các bit của các đồ vật đã in ra
                FOR(i,0,n-1) {

                        if(BIT(ck_mask,i)) {
                                cout << i + 1 << " ";
                        }
                }
                j --;
                cout << 0 <<  " ";
                mask = Tv[mask];
        }
   return 0;
```

```
}
/// Change Dream Magic Dr_Unicorn
```

4.4. Test trong thư mục đính kèm

2.5. Bài 5: Reverse

5.1. Đề bài:

(https://codeforces.com/contest/1234/problem/F)

Bạn được cung cấp một chuỗi S chỉ bao gồm đầu tiên 20 chữ cái Latinh viết thường (' a ', ' b ', ..., ' t ').

Nhớ lại rằng chuỗi con S [l ; r] S là chuỗi S_l S_{l+1} … S_r. Ví dụ: các chuỗi con của " magiccode " là " code ", " magic ", " gic ", " icc ", nhưng không phải là " made " và " gide ".

Bạn có thể thực hiện thao tác sau không quá một lần : chọn một số chuỗi con S [l ; r] và đảo ngược nó (tức là chuỗi S_l S_{l+1} … S_r trở thành S_r S_{r-1}… S_l).

Mục tiêu của bạn là tối đa hóa độ dài của chuỗi con tối đa của S bao gồm các ký tự riêng biệt (tức là duy nhất).

Chuỗi bao gồm các ký tự riêng biệt nếu không có ký tự nào trong chuỗi này xuất hiện nhiều hơn một lần. Ví dụ: chuỗi " abcde ", " arctg " và " minecraft " bao gồm các ký tự riêng biệt nhưng chuỗi " magiccode", " abacaba " không bao gồm các ký tự riêng biệt.

Input :

- Dòng duy nhất của đầu vào chứa một chuỗi S bao gồm không nhiều hơn 10^6 ký tự ' a ', ' b ', ..., ' t ' (đầu tiên 20 chữ cái Latinh viết thường).

Output :

- In một số nguyên - độ dài tối đa có thể có của chuỗi con tối đa của S bao gồm các ký tự riêng biệt sau khi đảo ngược không quá một chuỗi con của nó.

Examples:

Reverse.inp	**Reverse.out**
abacaba	3

5.2. Thuật toán hiệu quả nhất

- Bài này cho ta 1 lần lật ngược 1 chuỗi con trong xâu đấy. Ta có thế kiến 2 chuỗi con riêng biệt cách nhau trở thành 2 chuỗi con nối liền nhau tạo thành 1 chuỗi con lớn.
- Bài yêu câu sau tí nhất 1 thao tác lật ngược ta có được 1 chuỗi con có các kí tự riêng biệt và có độ dài lớn nhất.
- Từ nhận xét ta có thể biến đổi đề là tìm 2 chuỗi con kí tự khác nhau sao cho tổng độ dài của chúng lớn nhất
- Ta không quan trọng vị trí chuỗi con đó nên ta sẽ sử dụng BitMask
- Dp[Mask] độ dài lớn nhất của chuỗi con riêng biệt có kí tự là Một số bit bật trong mask Dễ dàng để QHĐ tạo ra Dp[Mask] đó
  + Khởi tạo Dp[mask] là độ dài chuỗi có kí tự riêng biệt bật trong mask
  + QHĐ Dp[Mask] độ dài lớn nhất của chuỗi con riêng biệt có kí tự là **Một Số** bit bật trong mask

```
for(int mask = 0 ; mask < (1 << 20) ; mask ++) {
        for(int j = 0 ; j < 20 ; j ++)  {
                if(BIT(mask, j))
                        Dp[mask] = max(Dp[mask], Dp[mask ^ (1 << j)]);
        }
}
```

- Sau khi QHĐ xong ta sẽ cập nhập kết quả
- Chọn các chuỗi con có kí tự riêng biệt (Dp[mask] == số lượng bit bật trong mask) và tìm chuỗi còn lại sao cho các kí tự 2 chuỗi riêng biệt cập nhật kết quả

```
for(int mask = 0 ; mask < (1 << 20) ; mask ++)  {
         if(Dp[mask] == slbit(mask))  {
                int re_mask = (MASK(20)-1) ^ mask;
                Ans = max(Ans, Dp[mask] + Dp[re_mask]);
        }
}
```

5.3. Codes

```
#include <bits/stdc++.h>
#define task "Reverse"
#define FOR(i,a,b) for(int i = (a); i <= (b); i ++)
#define MASK(i) (1LL << (i))
#define BIT(x, i) (((x) >> (i)) & 1) // kiểm tra bit thú i của số x
```

```
#define SET_ON(x, i) ((x) | MASK(i)) // bật bit thứ i của số x
#define SET_OFF(x, i) ((x) & ~MASK(i)) // tắt bit thứ i của số x
#define all(v) (v).begin(),(v).end()
#define ii pair<int, int>
#define jj pair<int, ii>
#define fi first
#define se second
#define int long long
using namespace std;
const int INF = (int)1e9;
const int mod = 1e9 + 7;
#define eps 1e-11

void IOS() {
   ios_base::sync_with_stdio(0);
   cin.tie(0);
   cout.tie(0);
}
void FRE() {
   //freopen("E:/MakeTest/Test.txt","r",stdin);
   freopen(task".inp", "r", stdin);
   freopen(task".out", "w", stdout);
}

const int N = 502;

string x;
int Dp[(1 << 20) + 5];
int Ok[22];

int slbit(int mask)
{
        return __builtin_popcount(mask);
}

main()
{
        IOS();
        FRE();

        cin >> x;
        for(int i = 0 ; i < x.size() ; i ++)
```

```
        {
                memset(Ok, 0 , sizeof(Ok));
                int mask = 0;
                for(int j = i ; j < x.size() ; j ++)
                {
                        if(!Ok[x[j] - 'a'])
                        {
                                Ok[x[j] - 'a'] = 1;
                        }
                        else
                        {
                                break;
                        }
                        mask |= (1 << (x[j] - 'a'));
                        Dp[mask] = slbit(mask);
                        // Khởi tạo Dp, mask có các bit bật là các kí tự trong đấy
                        // Dp[mask] lúc này là độ dài chuỗi có kí tự riêng biệt bật trong
mask
                }
        }

        // QHĐ như trong sol viết
        // để Dp[mask] trở thành độ dài lớn nhất của chuỗi con riêng biệt có kí tự là Một
Số bit bật trong mask
        for(int mask = 0 ; mask < (1 << 20) ; mask ++)
        {
                for(int j = 0 ; j < 20 ; j ++)
                {
                        if(BIT(mask, j))
                        {
                                Dp[mask] = max(Dp[mask], Dp[mask ^ (1 << j)]);
                        }
                }
        }
        int Ans = 0;
        for(int mask = 0 ; mask < (1 << 20) ; mask ++)
        {
                if(Dp[mask] == slbit(mask))
                {
                        // ta chọn chuỗi có các kí tự riêng biệt bật là mask
                        int re_mask = (MASK(20)-1) ^ mask;
                        // chọn chuỗi còn lạ có các kí tự riêng biệt bật là Một Số bit bật
```

```
trong re_mask
                        Ans = max(Ans, Dp[mask] + Dp[re_mask]);
                        // Cập nhập kết quả
                }
        }
        cout << Ans;
        return 0;
}
/// Ch/// Dream Magic Dr_Unicorn
```

5.4. Test trong thư mục đính kèm

III. MỘT SỐ BÀI TẬP ÁP DỤNG

3.1 BGAME (Đề thi chọn HSGQG ngày 1 năm hoạc 2016-2017)

➢ Sub1:

- QHĐ kết hợp với bitmask.
- Cố định cạnh i là max. Ta sẽ QHĐ từ đầu mút này đến đầu mút kia sao cho cạnh min là lớn nhất (Xét đồ thị chỉ gồm các cạnh ≤ max).

➢ Sub 2:

- Cây khung lớn nhất kết hợp với duyệt DFS
- Ta sẽ nối tất cả những cạnh lớn nhất để tạo thành một cây khung lớn nhất. Giả sử cạnh thêm vào là (u, v) thì max sẽ là cạnh lớn nhất trên đường đi u từ v trên cây. Nó sẽ tạo thành 1 chu trình từ u đến v

➢ Subtask 3:

- Cải tiến từ Subtask 2: Cây khung lớn nhất kết hợp với tìm cạnh lớn nhất trên đường đi từ u đến v bằng LCA

3.2 EFILL (Đề thi chọn HSGQG ngày 2 năm học 2018-2019)

Sub1: 50% số điểm ứng với m <=2000

- Sắp xếp lại dãy và lược bỏ các số trùng nhau đi.
- Sử dụng QHĐ bitmask
- Ta sẽ quản lý 10 phần tử liên tiếp nhau một
- Bitmask đánh dấu các phần tử đã chọn trước đây

  1001001011 là các số cách số đang xét lần lượt 10,7,4,2,1

3.3 LIGHT (Đề thi chọn HSGQG ngày2 năm học 2019-2020)

Sub1 : Sử dụng quy hoặc động bitmask.

- Bitmask thứ nhất lưu bóng nào bật, Bitmask thứ hai lưu bóng bật thì màu nào.
- QHĐ trạng thái bình thường.

3.4 Little Pony and Harmony Chest

(https://codeforces.com/problemset/problem/453/B)

Solution

- Nhận xét: Các số cần dùng từ nằm trong khoảng từ 2 cho đến 59
- Số lượng số cần dùng là 17. Vì vậy ta sẽ lưu vào bitmask
- Ta sẽ phân tích ra thừa số nguyên tố và khởi tạo sẵn một mảng B[K] – đánh dấu những thừa số nguyên tố xuất hiện trong K là bit bật
- Quy hoạch động:

$$solve(i, mask) = min(solve(i+1, mask | B[j]) + |j - A[i]|)$$

Với j:= 1 → 59 và mask & B[j]) = 0

3.5 CHNREST

(https://vn.spoj.com/problems/CHNREST/)

Solution

- Ta vẫn QHĐ bitmask như bình thường. Tuy nhiên thay vì ta dùng hệ nhị phân thì ở đây ta sẽ sử dụng hệ tam phân để lưu trữ

- Solve(i, mask) là số tiền phải trả ít nhất khi đã xét i món đầu tiên.

  Bit thứ j trong mask là số món người j đã ăn và yêu thích món đó.

Món thứ i được ăn khi và chỉ khi có ít nhất 1 người yêu thích món i và số món yêu thích người đó đã ăn nhỏ hơn 2.

3.6 MAUGIAO - The problem for kid

https://codeforces.com/group/FLVn1Sc504/contest/274822/problem/D

➢ Solution:

- Ta quy hoạch động 2 lần. 1 lần để tính tổng số tài sản lớn nhất tìm đc. Sau đó quy hoạch động thêm 1 lần nữa để tìm ra đc số cách có được tổng số tài sản như thế.
- Dp top-down tính tổng tài sản lớn nhất:
  Lưu ý: Ta nên đánh số mảng A bắt đầu từ 0 để khi dp bitmask sẽ ko bị thừa bộ nhớ.
  + Nếu x < N:
  solve(x, mask) = max(solve(x + 1, ONBIT(mask, i) + A[x][i]);
  (i = 0 ÷ N; bit thứ i của mask tắt)

  + Nếu x >= N:
  solve(x, mask) = 0;

  x : đang xét đến người con gái thứ x.
  mask : để lưu lại nhưng tiến sĩ nào đã đc ghép cặp vs 1 người con gái rồi (bit thứ i tắt nghĩa là tiến sĩ thứ i chưa đc ghép cặp; ngược lại nếu bit thứ i bật nghĩa là tiến sĩ thứ i đã đc ghép cặp).
- Dp top-down tính số cách có được tổng số tài sản max:
  + Nếu x < N :
  trace(x, mask) += trace(x + 1, ONBIT(mask, i));
  (i = 0 ÷ N; bit thứ i của mask tắt;
  solve(x + 1, ONBIT(mask, i)) + a[x][i] == solve(x, mask))

  + Nếu x >= N :
  trace(x, mask) = 1;

TÀI LIỆU THAM KHẢO

1. https://www.hackerearth.com/practice/algorithms/dynamic-programming/bit-masking/tutorial/?

fbclid=IwAR2RG8fGRbrn7FH2Ri_MAijUoFN7XCvQjBnskUMsFunlN69CeXDA3zRdcAw

2. https://laptrinhthidau.wordpress.com/2016/10/18/mot-so-ki-thuat-giai-bai-toan-bang-phuong-phap-quy-hoach-dong/?fbclid=IwAR11M8splBG5MTlQpYPIsg7Nv2OReQ4PFbWb1nTbh8tvyMRRfyaSZ4AXZu4

3. https://algorithmtips.blogspot.com/2013/06/bitmasks.html?fbclid=IwAR3JGzZqd4_NbC0_fC1gNC_KaLOkfK3grY9jF9-wXQB5dF4lXcTciEs1DQU

4. https://vnoi.info/problems/list/?tag=52&page=1&fbclid=IwAR3gSXszJcgNo3gmv5S3cBQWduqYxYJrBVyl197bfqREn_N_QM6TZNdNXmo

5. http://acmicpc-vietnam.github.io/2016/regional/solutions/ACMICPCRegionalVietnam2016-Editorial.pdf?fbclid=IwAR2nzbj-DN6b34-ZEuBYePRCE6Qlfwl9PuR7VJ3IBXyR6TLBJTqDdxcrfMM

6. https://www.geeksforgeeks.org/bitmasking-and-dynamic-programming-set-1-count-ways-to-assign-unique-cap-to-every-person/?ref=rp&fbclid=IwAR2E0xqEm9IPWA7UuGoh2GojK5BtqetfnnmMsZvtzeiSJbeQlAPoDRuBrbo

7. https://codeforces.com/blog/entry/18169?fbclid=IwAR2HRU8oYWs4V37OorMBTMvEii0dJxyLN9HZ7Bzen1ajl-EAo_R6PRLmpNE