Contents

1 語法

1.1 c++

```cpp
// c++ code
#include <bits/stdc++.h>
using namespace std;
int main(){
  std::ios::sync_with_stdio(false); // 加速
  return 0;
}

// struct宣告
struct s{
  int x[100];
  int y[100];
};

s num; //一組s
num.x[1]=1;
num.y[1]=2;

// sort
sort(v.begin(), v.end())      // array 不能用
sort(v, v+n)
sort(v, v+n, greater<int>()) // 大到小

sort(v, v+n, cmp)             // 自己寫比較序列
bool cmp(型態 a, 型態 b){
  return a > b;               // 大到小
}

// set
s.insert(x) //將x插入s中  O(log(n))
s.count(x)  //回傳x是否存在於s中()  O(log(n))
s.erase(x)  //刪除在s中的x  O(log(n))
s.clear()   //刪除s中所有元素 O(n)
s.empty()   //回傳是否為空  O(1)
s.size()    //回傳共有幾個元素  O(1)

map
insert(x) //將x這個pair插入map中  O(log(n))
count(x)  //回傳x這個key是否在map中 O(log(n))
erase(x)  //刪除在map中key為x的 O(log(n))

// vector
vector<int> v // 宣告
v.push_back(1) // 推入數字
v.pop_back()   // 拔出尾端數字
```

1.2 python

```python
# EOF
while True:
    try:
        '''
        你要執行的程式碼
        '''
    except EOFError:
        break

# 有規定終止條件
while True:
    if a==0:
        break

# 數學符號
a//=10 # 整除
a**b   # a^b

# 邏輯
a=True
b=False
print(a and b) #False
print(a or b)  #True

# scan
a = int(input())
n=list(input().split(' '))           #
    連續輸入一串用空格隔開的數字

for i in range(a):
    c, d = map(int, input().split()) # 連續輸入兩個數

# print
print('for is not a multiple of 11.'.format(a))
print(a+" and "+b+" sitting in the tree")
print('The parity of ',a,' is ',count,' (mod 2).')

# 標頭檔math
import math
math.gcd(a, b, c, d, e) # 最大公約數
math.lcm(a, b, c, d, e) # 最小公倍數
math.fabs(x)   # 絕對值
math.isqrt(n)  # 整數平方根
math.sqrt(x)   # 平方根
math.pow(x, y) # x^y

# count
c+=b.count("商店")  # 用在要計算好幾個字串時
c=b.count('1')     # 一次算出一串字串有幾個 '1'

# 進制轉換
a = bin(a)[2:] # 10 to 2
a = hex(a)[2:] # 10 to 16
a = oct(a)[2:] # 10 to 8

# 大小寫轉換
a.lower()
a.upper()

# 取長度
a.len()
```

2 字串

2.1 KMP

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

void KMP(string text, string pattern)
{
    int m = text.length();
    int n = pattern.length();

    // 如果模組沒東東
    if (n == 0)
    {
        cout << "The pattern occurs with shift 0";
        return;
    }

    // 如果文本的長度小於模組的長度
    if (m < n)
    {
        cout << "Pattern not found";
        return;
    }

    // next[i] 存儲下一個最佳部分匹配的索引
    int next[n + 1];

    for (int i = 0; i < n + 1; i++) {
        next[i] = 0;
    }

    for (int i = 1; i < n; i++)
    {
        int j = next[i];

        while (j > 0 && pattern[j] != pattern[i]) {
            j = next[j];
        }

        if (j > 0 || pattern[j] == pattern[i]) {
            next[i + 1] = j + 1;
        }
    }

    for (int i = 0, j = 0; i < m; i++)
    {
        if (text[i] ==
            pattern[j])//一樣如果+1j下一個檢查
        {
            if (++j == n) {
                cout << "The pattern occurs with
                    shift " << i - j + 1 << endl;
            }
        }
        else if (j > 0)
        {
            j = next[j];//把她休崩變回來
            i--;    // 還要回去啾啾
        }
    }
}

int main()
{
    string text = "ABCABAABCABAC";
    string pattern = "CAB";

    KMP(text, pattern);

    return 0;
}
```

3 數論

3.1 喵

```cpp
#include <iostream>
using namespace std;
int a[20];

int main() {
    int cases, target, sticks, num, tmp, i;
    bool flag;
    while (cin >> cases){
        while (cases--){
            cin >> target;
            cin >> sticks;
            for (int i = 0; i < sticks; i++){
                cin >> a[i];
            }
            num = 1;
            for (int i = 1; i < sticks; i++){
                num <<= 1;
                num++;
            }
            flag = false;
            for (int _i = 0; _i <= num; _i++){
                tmp = 0;
                i = _i;
                for (int j = 0; j < sticks; j++){
                    if (i & 1) tmp += a[j];
                    i >>= 1;
                }
                if (tmp == target){
                    flag = true;
                    break;
                }
            }
            if (flag) cout << "YES\n";
            else cout << "NO\n";
        }
    }
}
```

3.2 Fibonaccimal

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){

    int N;
    int Fibonacci[40] = {0, 1};//開始的兩個數
    int i;

    for(i = 2; i < 40; i++){
        Fibonacci[i] = Fibonacci[i - 1] + Fibonacci[i
            - 2];
    }

    scanf("%d", &N);

    while(N--){

        int num;
        bool flag = false;

        scanf("%d", &num);
        printf("%d = ", num);

        for(i = 39; i >= 2; i--){
            if(num >= Fibonacci[i]){
                num = num - Fibonacci[i];
                flag = true;
                printf("1");
            }
```

```
            else if(flag){
                printf("0");
            }
        }

        printf(" (fib)\n");
    }

    return 0;
}
```

3.3 LCM

```
int GCD(int num1,int num2)
{
  if(num2==0)
  {
   return num1;
  }

  return GCD(num2,num1%num2);
}

int LCM(int num1,int num2) //2個最小公倍數
{
  return((num1*num2)/GCD(num1,num2));
}

int LCM2(int num1,int num2,int num3) //3個最小公倍數
{
  return((num1*num2)/GCD((num1,num2),num3));
}

int main()
{
    cout<<GCD(6,3);
    cout<<LCM(6,3);
    cout<<LCM2(6,3,3);

    return 0;
}
```

3.4 LCS

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string str1, str2;
    short lcs[2][1000];
    while (cin >> str1 >> str2)
    {
        lcs[0][0] = str1[0] == str2[0];
        for (int j=1; j<str2.length(); j++) lcs[0][j]
            = max(lcs[0][j-1],
            short{str1[0]==str2[j]});
        for (int i=1; i<str1.length(); i++)
        {
            bool r = i & 1;
            lcs[r][0] = max(lcs[r^1][0],
                short{str1[i]==str2[0]});
            for (int j=1; j<str2.length(); j++)
                lcs[r][j] = str1[i]==str2[j] ?
                    lcs[r^1][j-1] + 1 :
                    max(lcs[r^1][j], lcs[r][j-1]);
        }
        cout <<
            lcs[(str1.length()-1)&1][str2.length()-1]
            << '\n';
    }
    return 0;
}
```

3.5 LPS

```
#include<bits/stdc++.h>
#include<iostream>
using namespace std;

int main(){
    string s;
    cin >> s;
    int maxlen=0, l, r;
    for(int i=0; i<s.length(); i++){
        //奇
        int x = 0;
        while((s[i-x]==s[i+x]) && (i-x>=0) &&
            (i+x<s.length())){//當找到一個中心點以其為中間然後左
            x++;
        }
        x--;
        if(2*x+1 >
            maxlen){//只有第一次會max==後後面就追不到那女孩
            maxlen = 2*x+1;//最大的
            l = i-x;//計算頭頭
            r = i+x;//計算尾巴
        }
        //偶
        x = 0;
        while( (s[i-x]==s[i+1+x]) && (i-x>=0) &&
            (i+1+x<s.length()) ){
            x++;
        }
        if(2*x > maxlen){
            maxlen = 2*x;
            l = i-x+1;
            r = i+x;
        }
    }
    cout << maxlen << '\n';
    cout << l+1 << ' ' << r+1 << '\n';

}
```

3.6 Pairty

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

int main() {
    int I, n;
    while (cin >> I) {
        if (I == 0) break;
        string B = "";
        n = I;
        int cnt = 0;
        while (n){
            cnt += (n & 1);
            B += '0' + (n & 1);
            n >>= 1;
        }
        reverse(B.begin(), B.end());
        cout << "The parity of " << B << " is " <<
            cnt << " (mod 2).\n";
    }
    return 0;
}
```

4 圖論

4.1 最短路徑 dijkstra

```cpp
// 邊權重皆為正數時使用
// 1.
    輸入有總點、總邊，接著輸入點,點,距離（權重）時使用
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

// 定義城市數量的上限
#define MAX_CITIES 100

// 定義無限大的距離
#define INF INT_MAX

// 城市數量、道路數量
int numCities, numRoads;

// 圖的鄰接矩陣表示法
vector<vector<int> > graph(MAX_CITIES,
    vector<int>(MAX_CITIES, INF));

//
    Dijkstra演算法，計算從指定城市出發到其他城市的最短路徑
void dijkstra(int startCity) {
    vector<int> dist(numCities, INF);
    vector<bool> visited(numCities, false);

    dist[startCity] = 0;

    for (int i = 0; i < numCities - 1; i++) {
        int u = -1;
        for (int j = 0; j < numCities; j++) {
            if (!visited[j] && (u == -1 || dist[j] <
                dist[u])) {
                u = j;
            }
        }

        visited[u] = true;

        for (int v = 0; v < numCities; v++) {
            if (!visited[v] && graph[u][v] != INF) {
                dist[v] = min(dist[v], dist[u] +
                    graph[u][v]);
            }
        }
    }

    // 輸出最短路徑結果
    cout << "從城市 " << startCity << "
        出發到其他城市的最短路徑如下：" << endl;
    for (int i = 0; i < numCities; i++) {
        if (i != startCity) {
            cout << "到城市 " << i << " 的最短距離為
                " << dist[i] << endl;
        }
    }
}

int main() {
    // 讀取城市數量和道路數量
    cin >> numCities >> numRoads;

    // 初始化圖的鄰接矩陣
    for (int i = 0; i < numRoads; i++) {
        int city1, city2, distance;
        cin >> city1 >> city2 >> distance;
        graph[city1][city2] = distance;
        graph[city2][city1] = distance; //
            因為是雙向道路
    }

    // 選擇起始城市，這裡以城市0為例
    int startCity = 0;

    // 執行Dijkstra演算法
    dijkstra(startCity);

    return 0;
}
```

4.2 DFS

```cpp
// 印出最快路徑（座標）
#include<bits/stdc++.h>
#define N 100
using namespace std;

int map[N][N], visited[N][N]={0};
typedef pair<int, int> p;
int n,m,found=0;
deque<p> path;

void dfs(int x, int y){
  if (found==1) return;
  visited[x][y]=1;
  path.push_back(make_pair(x,y));
  if (x==n-1 && y==m-1){
    found=1;
    cout<<"Path: ";
    while(!path.empty()){
      cout<<"("<<path.front().first<<","<<path.front().second<<
      path.pop_front();
      cout<<((path.empty())?"\n":"→");
    }
    cout<<endl;
    return;
  }
  if (x+1<n && visited[x+1][y]==0 && map[x+1][y]==0){
    dfs(x+1,y);
    path.pop_back();
  }
  if (y+1<m && visited[x][y+1]==0 && map[x][y+1]==0){
    dfs(x,y+1);
    path.pop_back();
  }
  if (x-1>=0 && visited[x-1][y]==0 && map[x-1][y]==0){
    dfs(x-1,y);
    path.pop_back();
  }
  if (y-1>=0 && visited[x][y-1]==0 && map[x][y-1]==0){
    dfs(x,y-1);
    path.pop_back();
  }
}

int main(){
  cin>>n>>m;
  for (int i=0; i<n; i++)
    for (int j=0; j<m; j++)
      cin>>map[i][j];
  dfs(0,0);
  if (found==0){
    cout<<"No routes accessible.\n";
  }
  return 0;
}
// 顯示最短距離
#include <iostream>
#include <utility>
#include <deque>
#define N 100
using namespace std;

int map[N][N], visited[N][N]={0};
typedef pair<int, int> p;
int n,m,dis=-2;
deque<p> path;

void dfs(int x, int y){
```

```
  visited[x][y]=1;
  path.push_back(make_pair(x,y));
  if (x==n-1 && y==m-1){
    if (dis==-1){
      dis=path.size()-1;
    }
    else {
      if (path.size()-1<dis) dis=path.size()-1;
    }
  }
  if (x+1<n && visited[x+1][y]==0 && map[x+1][y]==0){
    dfs(x+1,y);
    visited[x+1][y]=0;
    path.pop_back();
  }
  if (y+1<m && visited[x][y+1]==0 && map[x][y+1]==0){
    dfs(x,y+1);
    visited[x][y+1]=0;
    path.pop_back();
  }
  if (x-1>=0 && visited[x-1][y]==0 && map[x-1][y]==0){
    dfs(x-1,y);
    visited[x-1][y]=0;
    path.pop_back();
  }
  if (y-1>=0 && visited[x][y-1]==0 && map[x][y-1]==0){
    dfs(x,y-1);
    visited[x][y-1]=0;
    path.pop_back();
  }
}

int main(){
  cin>>n>>m;
  for (int i=0; i<n; i++)
    for (int j=0; j<m; j++)
      cin>>map[i][j];
  dfs(0,0);
  if (dis==-2)
    cout<<"No routes accessible.\n";
  else
    cout<<"Shortest distance: "<<dis<<endl;
  return 0;
}
```

4.3 merge sort

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

//做比較大小的部分
void merge(int arr[], int l, int m, int r, int size)
{
    int i = l;
    int j = m + 1;
    int k = l;

    /* create temp array */
    int temp[size];

    while (i <= m && j <= r) {
        if (arr[i] <= arr[j]) {
            temp[k] = arr[i];
            i++;
            k++;
        }
        else {
            temp[k] = arr[j];
            j++;
            k++;
        }
    }
    / Copy the remaining elements of first half, if
        there are any /
    while (i <= m) {
        temp[k] = arr[i];
        i++;
        k++;
    }

    / Copy the remaining elements of second half, if
        there are any /
    while (j <= r) {
        temp[k] = arr[j];
        j++;
        k++;
    }

    / Copy the temp array to original array /
    for (int p = l; p <= r; p++) {
        arr[p] = temp[p];
    }
}

//做分開陣列的部分
void mergeSort(int arr[], int l, int r, int size)
{
    if (l < r) {
        // 找中間點 ex:陣列五個元素0-4 2是中間點
        //陣列分成兩組 0-2/3-4兩個部分
        //舉0-2陣列來說 中間點是1
        //陣列再分成 0-1/2兩個部分
        int m = (l + r) / 2;

        / 遞迴第一和第二部分*/
        //(也就是不斷的分)
        mergeSort(arr, l, m, size);
        mergeSort(arr, m + 1, r, size);

        // merge
        //當我分到不能再分 比較陣列內數值 小的放前面
        merge(arr, l, m, r, size);
    }
}

int main()
{
    cout << "Enter size of array: " << endl;
    int size;
    cin >> size;
    int myarray[size];

    cout << "Enter " << size << " integers in any
        order: " << endl;
    for (int i = 0; i < size; i++) {
        cin >> myarray[i];
    }
    cout << "Before Sorting" << endl;
    for (int i = 0; i < size; i++) {
        cout << myarray[i] << " ";
    }
    cout << endl;
    mergeSort(myarray, 0, (size - 1), size); //
        mergesort(arr,left,right) called

    cout << "After Sorting" << endl;
    for (int i = 0; i < size; i++) {
        cout << myarray[i] << " ";
    }

    return 0;
}
```

4.4 quick sort

```
include <iostream>
using namespace std;
// quick sort sorting algorithm
int Partition(int arr[], int s, int e)
{
```

```
 int pivot = arr[e];
 int pIndex = s;

 for(int i = s;i<e;i++)
 {
     if(arr[i]<pivot)
     {
     int temp = arr[i];
     arr[i] = arr[pIndex];
     arr[pIndex] = temp;
    //swapping 也就是說如果當前數值比指標小
        他就移到最前面
    //也就是陣列0的位置
     pIndex++;
    //下一個比指標小的數值放進陣列1的位置
     }
 }

 int temp = arr[e];
 arr[e] = arr[pIndex];
 arr[pIndex] = temp;
 //比指標數值小的都去前面了
 //將指標放到目前計數到的陣列位置
 //那指標前都比她小 指標後都比他大
 return pIndex; //回傳給p值
}

void QuickSort(int arr[], int s, int e)
//s stand for start index
//e stand for end index also (size-1)
{
 if(s<e)
 {
 int p = Partition(arr,s, e);
 QuickSort(arr, s, (p-1));
// recursive QS call for left partition
//做陣列前半部分 因為都比指標小 去進行內部排序
 QuickSort(arr, (p+1), e);
// recursive QS call for right partition
 }
}

int main()
{

 int size=0;
 cout<<"Enter Size of array: "<<endl;
 cin>>size;
 int myarray[size];

 cout<<"Enter "<<size<<" integers in any order:
     "<<endl;
 for(int i=0;i<size;i++)
 {
 cin>>myarray[i];
 }
 cout<<"Before Sorting"<<endl;
 for(int i=0;i<size;i++)
 {
 cout<<myarray[i]<<" ";
 }
 cout<<endl;

 QuickSort(myarray,0,(size-1));  // quick sort called

 cout<<"After Sorting"<<endl;
 for(int i=0;i<size;i++)
 {
 cout<<myarray[i]<<" ";
 }

 return 0;
}
```

4.5 二分搜

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

int binarySearch(int arr[], int left, int right, int
    x) {
  while (left <= right) {
    int mid = left + (right - left) / 2;

    if (arr[mid] == x) {
      return mid;
    }
     else if (arr[mid] < x) {
      left = mid + 1;
    }
     else {
      right = mid - 1;
    }
  }

  return -1;
}

int main() {
  int myarr[10];
  int num;
  int output;

  cout << "Please enter 10 elements ASCENDING order"
      << endl;
  for (int i = 0; i < 10; i++) {
    cin >> myarr[i];
  }
  cout << "Please enter an element to search" << endl;
  cin >> num;

  output = binarySearch(myarr, 0, 9, num);

  if (output == -1) {
    cout << "No Match Found" << endl;
  } else {
    cout << "Match found at position: " << output <<
        endl;
  }

  return 0;
}
如果我們超過25頁我還可以再縮減程式區 這是比較完整的
Floyd
void floyd(){
        for(intk=0;k<n;k++){ //中間點
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    dp[i][j]=min(dp[i][j],dp[i][k]+dp[k][j]);
                    //經過中間點k的路徑是否小於原始路徑
                    //小於則更新 不小於則不變動
                    //窮舉所有鬆弛的可能
                }
            }
        }
    }
```

5 dp

5.1 階乘 1

```
'''
!注意！long long 存到21!就會爆掉

1.  要你輸出階乘
好懶，請你直接用python
'''
```

```python
a = 0
while True:
    try:
        a = int(input())
        sum = 1
        for i in range(1,a+1):
            sum*=i
        a = str(a)
        print(a+'!')
        print(sum)
    except EOFError:
        break
```

5.2 階乘 2

```cpp
/*
2.  要你輸出階乘最後一個非0的數字
用dp表格先存1-10000數字的階乘，
同時因為我們只關心最後一個非0的數字，
所以可以每次乘完一階就讓他進while迴圈裡%10，
把0都去掉，到while迴圈外後再把arr[i]%=10000，
只留下剩下可能會影響結果的數值部分。
*/
typedef long long ll;
ll arr[10000];
void s(){
    arr[0]=1;
    for(ll i = 1; i <= 10000; i++){
        arr[i] = arr[i-1]*(i+1);
        while (arr[i] % 10 == 0) {
            arr[i] /= 10;
        }
         arr[i] %= 1000000;
    }
}
```

5.3 階梯問題

```cpp
/*
1.  問從左上角走到右下角有幾種解法
- 此問題可分為(1)往下(2)往右，兩個走法。
*/
const int H = 8, W = 8;
int f[2][W];    //
    兩條陣列，儲存最近算出來的問題答案。

void staircase_walk()
{     // [Initial States]
    for (int j=0; j<W; ++j) f[0][j] = 1;

    // [Computation]
    for (int i=1; i<H; i++)
       for (int j=1; j<W; j++)
           // 只是多了 mod 2，
          // 外觀看起來就像兩條陣列輪替使用。
         f[i % 2][j] = f[(i-1) % 2][j] + f[i %
             2][j-1];

      // 輸出結果
      cout << "由(0,0)走到(7,7)有" << f[7 % 2][7] <<
          "種走法";
  //  cout << "由(0,0)走到(7,7)有" << f[(H-1) %
      2][W-1] << "種走法";
}
```

5.4 極值問題（格子有權重）

```cpp
const int H = 8, W = 8;
int a[H][W];
int f[H][W];

void staircase_walk()
{
  // [Initial States]
  f[0][0] = a[0][0];
  for (int i=1; i<H; i++)
    f[i][0] = f[i-1][0] + a[i][0];
  for (int j=1; j<W; j++)
    f[0][j] = f[0][j-1] + a[0][j];

  // [Computation]
  for (int i=1; i<H; i++)
    for (int j=1; j<W; j++)
      f[i][j] = max(f[i-1][j], f[i][j-1]) + a[i][j];

  // 輸出結果
  cout << "由(0,0)走到(7,7)的最小總和" << f[7][7];
//  cout << "由(0,0)走到(7,7)的最小總和" <<
    f[H-1][W-1];

  int h, w;
  while (cin >> h >> w)
    cout << "由(0,0)走到(h,w)的最小總和" << f[h][w];
}
```