# Contents



# 1 基礎

## 1.1 binarySearch

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

int binary_search(const vector<int> &data, int key)
{
    int low = 0;
    int high = data.size()-1;
    while (low <= high)
  {
        int mid = int((low + high) / 2);
        if (key == data[mid])
            return mid;
        else if (key > data[mid])
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}

int main()
{
    vector<int> data = {1, 9, 2, 7, 4, 10, 3, 8, 5,
        6};
    int key = 7;

    sort(data.begin(), data.end());

    int ret = binary_search(data, key);
    if (ret == -1)
        cout << "找不到\n";
    else
        cout << "找到索引值" << ret << "\n";
  //lower_bound(a, a + n, k);     //最左邊 ≥ k 的位置
  //upper_bound(a, a + n, k);     //最左邊 > k 的位置
  //upper_bound(a, a + n, k) - 1; //最右邊 ≤ k 的位置
  //lower_bound(a, a + n, k) - 1; //最右邊 < k 的位置
  //[lower_bound, upper_bound) //等於 k 的範圍
  //equal_range(a, a+n, k);
}
```

## 1.2 stringstream

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

int main()
{
  stingstream ss;
  int num = 1234;
  string output;

  ss << num;
  ss >> output; //integer to string

  string_to_int << ss;
  string_to_int >> num; //string to integer

  ss.str("");
  ss.clear(); //initialization

  return 0;
}
```

# 2 STL

## 2.1 map

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

int main()
{
  map<int, string> m = {{1, "Tom"}};
  m.insert(pair<int, string>(7, "Jack"));
  m[15] = "John";
  for (const auto& s : m) // map<int,
      string>::iterator it = m.begin()
  {
    cout << s.first << " " << s.second << endl;
  }
  m.erase(7);
  for (auto it = m.rbegin(); it!= m.rend(); it++)
  {
    cout << it->first << " " << it->second << endl;
  }
  m.clear();
  return 0;
}
```

# 3 圖論

## 3.1 並查集

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

#define N 10
int p[N], rank[N]; // rank -> tree height, sz ->
    group size
void init()
{
    for (int i = 0; i < N; i++)
    {
        p[i] = i;
        rank[i] = 1;
    }
}
int Find(int x)
{
    if (x == p[x])
        return x;
    return p[x] = Find(p[x]);
}
void Union(int a, int b) // tree height
{
    a = Find(a);
    b = Find(b);
```

```
    if (a == b)
        return;
    if (rank[a] < rank[b])
        p[a] = b;
    else if (rank[a] > rank[b])
        p[b] = a;
    else
    {
        p[a] = b;
        rank[a]++;
    }
}
// void Union(int a, int b) // group size
// {
//     a = Find(a);
//     b = Find(b);
//     if (a == b)
//         return;
//     if (sz[a] < sz[b])
//         swap(a, b);
//     sz[a] += sz[b];
//     p[b] = a;
// }

int main()
{
  init();
  for (int i = 0; i < N; i++)
  {
    int a, b;
    cin >> a >> b;
    Union(a, b);
  }
  return 0;
}
```

## 3.2 最小生成樹

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

#define N 10
int p[N], sz[N];
struct Edge
{
    int s, t, w;
    Edge(int s, int t, int w) : s(s), t(t), w(w) {}
    bool operator < (const Edge &rhs) const { return
        w < rhs.w; }
};
void init()
{
    for (int i = 1; i <= N; i++)
    {
        p[i] = i;
        sz[i] = 1;
    }
}
int Find(int x)
{
    if (x == p[x])
        return x;
    return p[x] = Find(p[x]);
}
void Union(int a, int b) // group size
{
    a = Find(a);
    b = Find(b);
    if (a == b)
        return;
    if (sz[a] < sz[b])
        swap(a, b);
    sz[a] += sz[b];
    p[b] = a;
}
int kruskal()
{
    int cost = 0;
    vector<Edge> E;
    init();
    for (int i = 0; i < N; i++)
    {
        int s, t, w;
        cin >> s >> t >> w;
        E.push_back(Edge(s,t,w));
    }
    sort(E.begin(), E.end());
    for (auto it : E)
    {
        it.s = Find(it.s);
        it.t = Find(it.t);
        if (it.s == it.t)
            continue;
        cost += it.w;
        Union(it.s, it.t);
    }
    return cost;
}

int main()
{
    init();
    int cost = kruskal();
    cout << cost << endl;
    return 0;
}
```

## 3.3 最長共同子序列-LCS

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

#define N 120
string strA , strB ;
int t[N*N] , d[N*N] , num[N*N] ; //t and d 是 LIS
    要用到
// d 用來記住 LIS 中此數字的前一個數字
// t 當前 LIS 的數列位置
// num 則是我們根據 strB 的字元生成數列，用來找出最長
    LIS 長度
map<char,vector<int>> dict ; //記住每個字串出現的
    index 位置

int bs(int l , int r , int v ){ //binary search
    int m ;
    while(r>l){
        m = (l+r) /2 ;
        if(num[v] > num[t[m]]) l = m+1 ;
        else if (num[v] < num[t[m]]) r = m ;
        else return m ;
    }
    return r ;
}

int lcs(){
    dict.clear() ; //先將 dict 先清空
    for(int i = strA.length()-1 ; i >= 0 ; i--)
        dict[strA[i]].push_back(i) ;
    // 將每個字串的位置紀錄並放入 vector 中，請記住 i
        = strA.length() -1 才可以達到逆續效果

    int k = 0 ; //紀錄生成數列的長度的最長長度
    for(int i = 0 ; i < strB.length() ; i++){ // 依據
        strB 的每個字元來生成數列
        for(int j = 0 ; j < dict[strB[i]].size() ;
            j++)
        //將此字元在 strA 出現的位置放入數列
            num[++k] = dict[strB[i]][j] ;
    }
```

```
    if(k==0) return 0 ; //如果 k = 0
        就表示他們沒有共同字元都沒有於是就直接輸出 0

    d[1] = -1 , t[1] = 1 ; //LIS init
    int len = 1, cur ; // len 由於前面已經把 LCS = 0
        的機會排除，於是這裡則從 1 開始

    // 標準的 LIS 作法，不斷嘗試將 LCS 生長
    for(int i = 1 ; i <= k ; i++ ){
        if(num[i] > num[t[len]]) t[++len] = i , d[i]
            = t[len-1] ;
        else{
            cur = bs(1,len,i);
            t[cur] = i ;
            d[i] = t[cur-1];
        }

        //debug
       // for(int i = 1 ; i <= k ; i++)
       //     cout << num[t[i]] << ' ' ;
       // cout << '\n' ;
    }
    return len ;

}

int main()
{
    getline(cin, strA);
    getline(cin, strB);
    cout << lcs() << endl;
    return 0;
}
```

### 3.4 Floyd-Warshall

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

#define N 10+5

int main()
{
  int G[N][N];
  for (int i = 0; i < N; i++)
  {
    for (int j = 0; j < N; j++)
    {
      cin >> G[i][j];
      G[j][i] = G[i][j];
    }
  }
  for (k = 0; k < n; k++)
      for (i = 0; i < n; i++)
          for (j = 0; j < n; j++)
              w[i][j] = w[j][i] = min(w[i][j],
                  w[i][k] + w[k][j]);
  return 0;
}
```

## 4 數學

### 4.1 質數表

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

#define maxn 100+5
vector<int> p;
bitset<maxn> is_notp;
void PrimeTable(int n)
{
    is_notp.reset();
    is_notp[0] = is_notp[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        if (is_notp[i])
            continue;
        p.push_back(i);
        for (int j = i * i; j <= n; j += i)
        {
            is_notp[j] = 1;
        }
    }
}

int main()
{
  PrimeTable(100);
  cout << is_notp[3];
  return 0;
}
```

### 4.2 gcd-lcm

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

int gcd(int a, int b)
{
  if (b == 0) return a;
  return gcd(b, a%b);
}

int lcm(int a, int b)
{
  if (a < b) swap(a, b);
  return a / gcd(a, b) * a;
}

int main()
{
  int a, b;
  cin >> a >> b;
  cout << gcd(a, b) << endl << lcm(a, b) << endl;
  return 0;
}
```

## 5 anngood

### 5.1 bfs

```
#include<bits/stdc++.h>
using namespace std;
int a[500][500]={0};
int used[500][500] = {0};

int moveI[4]={-1, 0, 1, 0};
int moveJ[4]={0, 1, 0, -1};
struct node{
    int I, J, now;
    node(int I, int J, int now):I(I), J(J),
        now(now){};
};
int main(){
    int N;
    cin>>N;
    while(N--){
        memset(a, 0, sizeof(a));
        memset(used, 0, sizeof(used));
        int n, m, sn, sm, en, em;
        cin>>n>>m>>sn>>sm>>en>>em;

```

```
        for(int i=0; i < n+2; i++){ //建牆壁 若不用可跳過
            a[i][0] = 1;
            a[i][m+1] = 1;
        }
        for(int i=0; i < m+2; i++){
            a[0][i] = 1;
            a[n+1][i] = 1;
        }

        for(int i=1;i<=n;i++){ // 用char輸入 因為輸入無空格
            for(int j=1;j<=m;j++){
                char t;
                cin >> t;
                a[i][j] = t-'0';
            }
        }

        queue<node> q;
        q.push(node(sn, sm, 1));  //定義起點為1步
        int flag = 0;
        while(!q.empty()){
            node F=q.front(); //取最優先那個
            q.pop();   //取完丟掉
            if(F.I == en && F.J == em){
                flag = 1;
                cout << F.now << '\n';
                break;
            }
            for(int i = 0; i < 4; i++){
                int nowi = F.I+moveI[i];   // 找I上下左右
                int nowj = F.J+moveJ[i];   // 找J上下左右
                if(a[nowi][nowj] == 0 &&
                    used[nowi][nowj] == 0){    // 如果可走且沒走過
                    used[nowi][nowj]=1;
                    q.push(node(nowi, nowj,
                        F.now+1));  // 放到queue裡面(最不優先)
                }
            }
        }

        if(!flag) cout << 0 << '\n';
            //如果找不到路輸出0;
    }
    return 0;
}
```

## 5.2 bwt

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXSTR 1000
#define MARKER '$'
#define NUM_ALPHA 26

int comp(const void *s, const void *t) {
    return strcmp(*(char**)s, *(char**) t);    /* 注意char的用法 */
}

/* the last char of s is not MARKER */
char* bwtEncode(char *s, const int n) {
    char *L;
    char **M;
    int i, j, l;
    if ((M = (char**) malloc(sizeof(char*) * n)) == NULL || \
        (L = (char*) malloc(sizeof(char) * (n + 2))) == NULL) {
        fputs("Error: out of space!\n", stderr);
        exit(1);
    }
    for (i = 0; i < n; i++) {
        if ((M[i] = (char*) malloc(sizeof(char) * (i + 2))) == NULL) {   /* 只須保存開頭到標記的那一部份字母串，這樣總共可以省約
            */
            fputs("Error: out of space!\n", stderr);
            exit(1);
        }
        for (j = 0; j < i + 1; j++)
            M[i][j] = s[n - 1 - i + j];   /* 這裡的字母串沒有存MARKER */
        M[i][i + 1] = '\0';
    }
    qsort(M, n, sizeof(M[0]), comp);   /* 對旋轉後的多個字母串排列 */
    for (i = 0, L[0] = s[n - 1]; i < n; i++) {
        if ((l = strlen(M[i])) < n)
            L[i + 1] = s[n - 1 - l];
        else
            L[i + 1] = MARKER;
    }
    L[n + 1] = '\0';
    for (i = 0; i < n; i++)
        free(M[i]);
    free(M);
    return L;
}

char* bwtDecode(char *L, const int n) {
    int i;
    int *a, *b;
    char *r;   /* original string. */
    int pos;
    if ((a = (int*) calloc(NUM_ALPHA + 1, sizeof(int))) == NULL || \
        (b = (int*) calloc(n, sizeof(int))) == NULL || \
        (r = (char*) malloc(sizeof(char) * (n + 1))) == NULL) {
        fputs("Error: out of space!\n", stderr);
        exit(1);
    }
    for (i = 0; i < n; i++) {  /* L列中每種字母的個數 */
        if (L[i] == MARKER)
            a[0]++;
        else
            a[L[i] - 'a' + 1]++;
    }
    for (i = 1; i < NUM_ALPHA + 1; i++) {   /* F列中排在每種字母前面的其他字母的個數 */
        a[i] += a[i - 1];
    }
    for (i = 0; i < n; i++) {    /* L列中每個字母跳轉到F列中的位置 */
        if (L[i] == MARKER)
            b[i] = 0;
        else
            b[i] = a[L[i] - 'a']++;
    }
    for (i = 0, pos = 0; i < n; i++) {
        r[n - 1 - i] = L[pos];
        pos = b[pos];
    }
    r[n] = '\0';
    free(a);
    free(b);
    return r;
}

int main(void) {
    char s[MAXSTR];
```

```
    char *L;
    int n;
    char *r;
    printf("input str: ");
    fgets(s, MAXSTR - 1, stdin);
    n = strlen(s);
    if (s[n - 1] == '\n')
        s[--n] = '\0';
    L = bwtEncode(s, n);
    printf("The L column: %s\n", L);
    r = bwtDecode(L, ++n);
    printf("The original str: %s\n", r);
    free(L);
    free(r);
}
```