Contents

1 basic

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 graph

2.1 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;
}
```

2.2 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;
}
```

2.3 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;
}
```

2.4 Floyd-Warshall

```cpp
#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;
}
```

3 Section1

3.1 basic

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

int main() {
    // test comment
    cout << "test string\n";
}
```

4 Section2

4.1 thm

- 中文測試
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$