Contents

1 Data Structure 1
1.1 BIT . . . 1
1.2 Segment tree . . . 1
1.3 Trie . . . 1

2 Dynamic Programming 1
2.1 Josephus . . . 1
2.2 LCS . . . 1
2.3 LIS . . . 2

3 Graph 2
3.1 Dijkstra . . . 2
3.2 Floyd Warshall . . . 2
3.3 SPFA . . . 2
3.4 Kruskal . . . 2
3.5 KM . . . 3
3.6 Dicnic . . . 3

4 Math 4
4.1 Inversion . . . 4

5 Function 4
5.1 strstr . . . 4
5.2 substr . . . 4
5.3 map set . . . 4
5.4 vector . . . 5
5.5 setprecision . . . 5
5.6 GCD LCM . . . 5
5.7 reverse . . . 5
5.8 CHAR . . . 5
5.9 sort . . . 5
5.10 struct . . . 5
5.11 deque . . . 5

1 Data Structure

1.1 BIT

```cpp
#define lowbit(k) (k & -k)
void add(vector<int> &tr, int id, int val) {
    for (; id <= n; id += lowbit(id)) {
        tr[id] += val;
    }
}
int sum(vector<int> &tr, int id) {
    int ret = 0;
    for (; id >= 1; id -= lowbit(id)) {
        ret += tr[id];
    }
    return ret;
}
```

1.2 Segment tree

```cpp
int dfs(int lef, int rig){
    if(lef + 2 == rig){
        if(num[lef] > num[rig-1]){
            return lef;
        }
        else{
            return rig-1;
        }
    }
    int mid = (lef + rig)/2;
    int p1 = dfs(lef, mid);
    int p2 = dfs(mid, rig);
    if(num[p1] > num[p2]){
        return p1;
    }
    else{
        return p2;
    }
}
```

1.3 Trie

```cpp
const int MAXL = ; // 自己填
const int MAXC = ;
struct Trie {
    int nex[MAXL][MAXC];
    int len[MAXL];
    int sz;
    void init() {
        memset(nex, 0, sizeof(nex));
        memset(len, 0, sizeof(len));
        sz = 0;
    }
    void insert(const string &str) {
        int p = 0;
        for (char c : str) {
            int id = c - 'a';
            if (!nex[p][id]) {
                nex[p][id] = ++sz;
            }
            p = nex[p][id];
        }
        len[p] = str.length();
    }
    vector<int> find(const string &str, int i) {
    int p = 0;
    vector<int> ans;
    for (; i < str.length(); i++) {
        int id = str[i] - 'a';
        if (!nex[p][id]) {
            return ans;
        }
        p = nex[p][id];
        if (len[p]) {
            ans.pb(len[p]);
        }
    }
    return ans;
    }
};
```

2 Dynamic Programming

2.1 Josephus

```cpp
int josephus (int n, int k) {
    // 有 n 個人圍成一圈，每 k 個一次
    return n > 1 ? (josephus(n-1 , k) + k) % n : 0;
}
// 回傳最後一人的編號, 0 index
```

2.2 LCS

```cpp
int LCS(string s1, string s2) {
    int n1 = s1.size(), n2 = s2.size();
    int dp[n1 + 1][n2 + 1];
    memset(dp, 0, sizeof(dp));
    // dp[i][j] = s1的前 i 個字元和 s2 的前 j 個字元
    for (int i = 1; i <= n1; i++) {
        for (int j = 1; j <= n2; j++) {
            if (s1[i - 1] == s2[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            }
            else {
                dp[i][j] = max(dp[i - 1][j], dp[i][j
                    - 1]);
            }
        }
    }
    return dp[n1][n2];
}
```

2.3 LIS

```cpp
int LIS(vector<int> &a) {
    vector<int> s;
    for (int i = 0; i < a.size(); i++) {
        if (s.empty() || s.back() < a[i]) {
            s.push_back(a[i]);
        }
        else {
            *lower_bound(s.begin(), s.end(), a[i]) =
                a[i];
        }
    }
    return s.size();
}
```

3 Graph

3.1 Dijkstra

```cpp
int M;  // number of vertex
int N;  // number of edge
int s, t, w;

struct Edge{

    int t, w;

    bool operator < (const Edge &rhs) const {
        return w > rhs.w;
    }
};

int dis[100001];
vector<Edge> G[100001];

void Dijkstra(int s){

    for(int i = 0; i <= M; i++){
        dis[i] = 1e9;
    }

    dis[s] = 0;
    priority_queue<Edge> pq;

    pq.push({s, 0});

    while(!pq.empty()){

        Edge now = pq.top();
        pq.pop();

        if(now.w > dis[now.t]){
            continue;
        }

        // relax
        for(Edge e: G[now.t]){
            if(dis[e.t] > now.w + e.w){
                dis[e.t] = now.w + e.w;
                pq.push({e.t, dis[e.t]});
            }
        }
    }
}
```

3.2 Floyd Warshall

```cpp
void floyd_warshall(){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            G[i][j] = INF;
        }
        G[i][i] = 0;
    }
    for (int k = 0; k < n; k++){      // 嘗試每一個中繼點
        for (int i = 0; i < n; i++){ // 計算每一個i點與每一個j點
            for (int j = 0; j < n; j++){
                G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
            }
        }
    }
}
```

3.3 SPFA

```cpp
bool SPFA(int s){
    // 記得初始化這些陣列
    int cnt[1000+5], dis[1000+5];
    bool inqueue[1000+5];
    queue<int> q;

    q.push(s);
    dis[s] = 0;
    inqueue[s] = true;
    cnt[s] = 1;
    while(!q.empty()){
        int now = q.front();
        q.pop();
        inqueue[now] = false;

        for(auto &e : G[now]){
            if(dis[e.t] > dis[now] + e.w){
                dis[e.t] = dis[now] + e.w;
                if(!inqueue[e.t]){
                    cnt[e.t]++;
                    if(cnt[e.t] > m){
                        return false;
                    }
                    inqueue[e.t] = true;
                    q.push(e.t);
                }
            }
        }
    }
    return true;
}
```

3.4 Kruskal

```cpp
struct Edge{
    int u, v, w;
    // 用權重排序 由大到小
    bool operator < (const Edge &other) const{
        return w > other.w;
    }
}edge[maxn];
// disjoint set
int find(int x){
  if(parent[x] < 0){
    return x;
  }
  else{
    return parent[x] = find(parent[x]);
  }
}
void unite(int a, int b){
  a = find(a);
  b = find(b);

  if(a != b){
    if(parent[a] < parent[b]){
```

```
      parent[a] += parent[b];
      parent[b] = a;
    }
    else{
      parent[b] += parent[a];
      parent[a] = b;
    }
  }
}
void kruskal(){
    memset(parent, -1, sizeof(parent));
    sort(edge, edge + m);
    int i, j;
    for(i = 0, j = 0; i < n - 1 && j < m; i++){
        // 如果 u 和 v 的祖先相同, 則 j++
            (祖先相同代表會產生環 所以不要)
        while(find(edge[j].u) == find(edge[j].v)) j++;
        // 若部會產生環 則讓兩點之間產生橋
            (連接兩顆子生成樹)
        unite(edge[j].u, edge[j].v);
        j++;
    }
}
```

3.5 KM

```
const int X = 50;    // X的點數，等於Y的點數
const int Y = 50;    // Y的點數
int adj[X][Y];       // 精簡過的adjacency matrix
int lx[X], ly[Y];    // vertex labeling
int mx[X], my[Y];    //
    X各點的配對對象、Y各點的配對對象
int q[X], *qf, *qb; // BFS queue
int p[X];            // BFS
    parent，交錯樹之偶點，指向上一個偶點
bool vx[X], vy[Y];   // 記錄是否在交錯樹上
int dy[Y], pdy[Y];   // 表格

void relax(int x){ // relaxation
    for (int y=0; y<Y; ++y)
        if (adj[x][y] != 1e9)
            if (lx[x] + ly[y] - adj[x][y] < dy[y]){
                dy[y] = lx[x] + ly[y] - adj[x][y];
                pdy[y] = x; //
                    記錄好是從哪個樹葉連出去的
            }
}
void reweight(){ // 調整權重、調整表格
    int d = 1e9;
    for (int y=0; y<Y; ++y) if (!vy[y]) d = min(d,
        dy[y]);
    for (int x=0; x<X; ++x) if ( vx[x]) lx[x] -= d;
    for (int y=0; y<Y; ++y) if ( vy[y]) ly[y] += d;
    for (int y=0; y<Y; ++y) if (!vy[y]) dy[y] -= d;
}
void augment(int x, int y){ // 擴充路徑
    for (int ty; x != -1; x = p[x], y = ty){
        ty = mx[x]; my[y] = x; mx[x] = y;
    }
}
bool branch1(){ // 延展交錯樹：使用既有的等邊
    while (qf < qb)
        for (int x=*qf++, y=0; y<Y; ++y)
            if (!vy[y] && lx[x] + ly[y] == adj[x][y]){
                vy[y] = true;
                if (my[y] == -1){
                    augment(x, y);
                    return true;
                }
                int z = my[y];
                *qb++ = z; p[z] = x; vx[z] = true;
                    relax(z);
            }
    return false;
}
bool branch2(){ // 延展交錯樹：使用新添的等邊
    for (int y=0; y<Y; ++y){
        if (!vy[y] && dy[y] == 0){
            vy[y] = true;
            if (my[y] == -1){
                augment(pdy[y], y);
                return true;
            }
            int z = my[y];
            *qb++ = z; p[z] = pdy[y]; vx[z] = true;
                relax(z);
        }
    }
    return false;
}
int Hungarian(){
    // 初始化vertex labeling
    //  memset(lx, 0, sizeof(lx));  // 任意值皆可
    memset(ly, 0, sizeof(ly));
    for (int x=0; x<X; ++x)
        for (int y=0; y<Y; ++y)
            lx[x] = max(lx[x], adj[x][y]);

    // X側每一個點，分別建立等邊交錯樹。
    memset(mx, -1, sizeof(mx));
    memset(my, -1, sizeof(my));
    for (int x=0; x<X; ++x){
        memset(vx, false, sizeof(vx));
        memset(vy, false, sizeof(vy));
        memset(dy, 0x7f, sizeof(dy));
        qf = qb = q;
        *qb++ = x; p[x] = -1; vx[x] = true; relax(x);
        while (true){
            if (branch1()) break;
            reweight();
            if (branch2()) break;
        }
    }
    // 計算最大權完美匹配的權重
    int weight = 0;
    for (int x=0; x<X; ++x)
        weight += adj[x][mx[x]];
    return weight;
}
```

3.6 Dicnic

```
// Maximum Flow
const int V = 100, E = 1000;
int adj[V]; // adjacency lists，初始化為-1。
struct Element {int b, r, next;} e[E*2];
int en = 0;
void addedge(int a, int b, int c){
    e[en] = (Element){b, c, adj[a]}; adj[a] = en++;
    e[en] = (Element){a, 0, adj[b]}; adj[b] = en++;
}
int d[V];        // 最短距離
bool visit[V];  // BFS/DFS visit record
int q[V];        // queue
int BFS(int s, int t){ // 計算最短路徑，求出容許圖
    memset(d, 0x7f, sizeof(d));
    memset(visit, false, sizeof(visit));
    int qn = 0;
    d[s] = 0;
    visit[s] = true;
    q[qn++] = s;

    for (int qf=0; qf<qn; ++qf){
        int a = q[qf];
        for (int i = adj[a]; i != -1; i = e[i].next){
            int b = e[i].b;
            if (e[i].r > 0 && !visit[b]){
                d[b] = d[a] + 1;
                visit[b] = true;
```

```
                q[qn++] = b;
                if (b == t) return d[t];
            }
        }
    }
    return V;
}
int DFS(int a, int df, int s, int t){ //
    求出一條最短擴充路徑，並擴充流量
    if (a == t) return df;
    if (visit[a]) return 0;
    visit[a] = true;
    for (int i = adj[a]; i != -1; i = e[i].next){
        int b = e[i].b;
        if (e[i].r > 0 && d[a] + 1 == d[b]){
            int f = DFS(b, min(df, e[i].r), s, t);
            if (f){
                e[i].r -= f;
                e[i^1].r += f;
                return f;
            }
        }
    }
    return 0;
}
int dinitz(int s, int t){
    int flow = 0;
    while (BFS(s, t) < V)
        while (true){
            memset(visit, false, sizeof(visit));
            int f = DFS(s, 1e9, s, t);
            if (!f) break;
            flow += f;
        }
    return flow;
}
```

4 Math

4.1 Inversion

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

int n;  // number of integer
int a[1000001];
int buf[1000001];

long long Inversion(int left, int right){

    if (right - left <= 1){ // finish split
        return 0;
    }

    int middle = (right + left) / 2;
    long long ans = Inversion(left, middle) +
        Inversion(middle, right);
    int i = left, j = middle, k = left;

    while (i < middle || j < right){
        if (i >= middle){
            buf[k] = a[j++];
        }
        else if (j >= right){
            buf[k] = a[i++];
        }
        else{
            if (a[i] <= a[j]){
                buf[k] = a[i++];
            }
            else{
                buf[k] = a[j++];
                ans += middle - i;
            }
        }
        k++;
    }

    for (int k = left; k < right; k++){
        a[k] = buf[k];
    }

    return ans;
}

int main(){

    while(cin >> n && n){

        memset(a, 0, sizeof(a));
        memset(buf, 0, sizeof(buf));

        for(int i = 0; i < n; i++){
            cin >> a[i];
        }

        cout << Inversion(0, n) << endl;
    }

    return 0;
}
```

5 Function

5.1 strstr

```
#include <stdio.h>
#include <string.h>

int main(){
char * c;
char str1[1005], str2[1005];
scanf("%s %s", str1, str2);
c = strstr(str1, str2);
if (c != NULL){
    printf("Yes\n");
}
else printf("No\n");
}
// Input : Hello eLl
// Output : No
```

5.2 substr

```
int main(){
    string str; //abcdef
    cin >> str;
    string tmp;
    tmp = str.substr(0, 2); //ab
    str = str.substr(2); //cdef
    cout << tmp << " " << str;
    return 0;
}
```

5.3 map set

```
.begin( ) // Return iterator to beginning
.end( ) // Return iterator to end
.empty( ) // 檢查是否為空
.size( )  // 回傳大小
mp.insert(pair<char,int>('a',100))
st.insert(100) // 插入key、value
.erase( ) // 刪掉指定key和他的value
```

```
.clear( ) // 清空整個 map
m.find( )
cout << "a => " << mymap.find('a')->second << endl;
    // 找出 map 裡 key
        有沒有在裡面，如果有的話會回傳元素所在的iterator，沒有則回傳end
s.count() // 返回某個值元素在set的個數
while( !mymap.empty()){
    cout << mymap.begin()->first << " => " <<
        mymap.begin()->second << endl;
    mymap.erase(mymap.begin());
}
for (auto it = mymap.begin(); it != mymap.end(); ++it)
    cout << it->first << " => " << it->second << endl;
```

5.4 vector

```
v.erase(v.begin() + 5) //拿掉第六個數
v.erase (v.begin(), v.begin() + 3); //拿掉前三個數
```

5.5 setprecision

```
// 將數字的小數部分設定為固定長度
cnt = 3.5555;
cout << fixed << setprecision(3) << cnt ;
// output : 3.555
```

5.6 GCD LCM

```
int gcd(int a, int b){
    return (b == 0 ? a : gcd(b, a % b));
}
int lcm(int a, int b){
    return a * b / gcd(a, b);
}

/* 輾轉相除法 - 求兩數是否互質
如果兩數互質 最終結果其中一方為0時 另一方必為1
若兩數有公因數 最終結果其中一方為0時 另一方必不為1 */
while ( ( num1 %= num2 ) != 0 && ( num2 %= num1 ) !=
    0 );
```

5.7 reverse

```
int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
reverse(a, a+5) // 轉換0~5

vector<int> v;
reverse(v.begin(), v.end());

string str = "123";
reverse(str.begin(), str.end());
cout << str << endl; //321
```

5.8 CHAR

```
isdigit()
isalnum() //判斷字母 || 數字
isalpha()
islower()
isupper()
isblank() //判斷是否為空格，或者 tab 健制表符，即
    space 和 \t
toupper()
tolower()
```

5.9 sort

```
priority_queue<int, vector<int>, less<int>> //大到小
priority_queue<int, vector<int>, greater<int>>
    //小到大

int arr[] = {4, 5, 8, 3, 7, 1, 2, 6, 10, 9};
    sort(arr, arr+10);

vector<int> v;
sort(v.begin(), v.end()); //小到大

int cmp(int a, int b){
    return a > b;
}
sort(v.begin(), v.end(), cmp); //大到小
```

5.10 struct

```
struct area{
    int a, b;
    bool operator<(const area rhs) const{
        return a > rhs.a || ( a == a && b > rhs.b);
    }
    bool operator!=(const area rhs) const{
        return a != rhs.a || b != rhs.b;
    }
};
```

5.11 deque

```
deque <int> que;
que.push_back(10);
que.push_front(20);
que.front()
que.back()
que.pop_front()
que.pop_back()
cout << "Element at position 2 : " << que.at(2) <<
    endl;
```