

# Contents

1	Basic	1
1.1	.vimrc	1
2	Dynamic Programming	1
2.1	0/1 Knapsack_problems	1
2.2	Complete_Knapsack_problems	1
2.3	Longest Common Subsequence(LCS)	1
2.4	Longest increasing common sequence(LICS)	1
2.5	Longest Increasing Subsequence(LIS)	2
2.6	Longest Palindromic Subsequence(LPS)	2
3	Graph Theory	2
3.1	Lowest Common Ancestor(LCA)	2
3.2	bellman-ford	2
3.3	dijkstra	2
3.4	Topological	3
4	Algorithm	3
4.1	Ternary Search	3
5	Number Theory	3
5.1	質數篩法 Sieve of Eratosthenes	3
6	Data Structure	3
6.1	Disjoint Set Union-Find	3
6.2	Segment Tree	3
7	String	4
7.1	Suffix Array	4
7.2	Suffix Array LCP	4
7.3	KMP algorithm	4
7.4	Manacher's algorithm	4
7.5	Z-algorithm	5
8	離散化 Discretization	5
8.1	Vector ( $O(N\log N)$ )	5
8.2	Map + Set ( $O(N\log N)$ )	5

## 1 Basic

### 1.1 .vimrc

```

1 filetype indent on
2 syntax enable
3 set nu
4 set cursorline
5 set ts=2 sts=2 sw=2 et ai
6 set mouse=a
7 set wrap
8 set showcmd
9 set backspace=indent,eol,start
10
11 inoremap ( (<ESC>i
12 inoremap [ [<ESC>i
13 inoremap {<CR> {<CR><ESC>ko

```

## 2 Dynamic Programming

### 2.1 0/1 Knapsack\_problems

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int f[1000]={0};
4 int n=0, m=0;
5 int main(){
6     cin >> n >> m;
7     for (int i = 1; i <= n; i++){
8         int price = 0, value = 0;
9         cin >> price >> value;
10        for (int j = m; j >= price; j--){
11            if (f[j-price]+value>f[j]){
12                f[j]=f[j-price]+value;
13            }
14        }

```

```

15    }
16    cout << f[m] << endl;
17    return 0;
18 }

```

### 2.2 Complete\_Knapsack\_problems

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int f[1000]={0};
4 int n=0, m=0;
5 int main(){
6     cin >> n >> m;
7     for (int i=1; i<=n; i++){
8         int price=0, value=0;
9         cin >> price >> value;
10        for (int j=price; j<=m; j++){
11            if (f[j-price]+value>f[j]){
12                f[j]=f[j-price]+value;
13            }
14        }
15    }
16    cout << f[m] << endl;
17    return 0;
18 }

```

### 2.3 Longest Common Subsequence(LCS)

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int dp[1001][1001];
5 int lcs(const string &s, const string &t){
6     int m = s.size(), n = t.size();
7     if (m == 0 || n == 0){
8         return 0;
9     }
10    for(int i = 0; i <= m; ++i){
11        dp[i][0] = 0;
12    }
13    for(int j = 1; j <= n; ++j){
14        dp[0][j] = 0;
15    }
16    for(int i = 0; i < m; ++i){
17        for (int j = 0; j < n; ++j){
18            if(s[i] == t[j]){
19                dp[i+1][j+1] = dp[i][j]+1;
20            }else{
21                dp[i+1][j+1] = max(dp[i+1][j],
22                                   dp[i][j+1]);
23            }
24        }
25    }
26    return dp[m][n];

```

### 2.4 Longest increasing common sequence(LICS)

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int a[100] = {0};
4 int b[100] = {0};
5 int f[100] = {0};
6 int n = 0, m = 0;
7 int main(){
8     cin >> n;
9     for(int i = 1; i <= n; i++){
10        cin >> a[i];
11    }
12    cin >> m;

```

```

13     for(int i = 1; i <= m; i++){
14         cin >> b[i];
15     }
16     for(int i = 1; i <= n; i++){
17         int k = 0;
18         for (int j = 1; j <= m; j++){
19             if(a[i] > b[j] && f[j] > k){
20                 k = f[j];
21             }else if(a[i] == b[j] && k + 1 > f[j]){
22                 f[j] = k + 1;
23             }
24         }
25     }
26     int ans=0;
27     for(int i = 1; i <= m; i++){
28         if(f[i] > ans){
29             ans = f[i];
30         }
31     }
32     cout << ans << endl;
33     return 0;
34 }

```

## 2.5 Longest Increasing Subsequence(LIS)

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int n=0;
4 int a[100]={0}, f[100]={0}, x[100]={0};
5 int main(){
6     cin >> n;
7     for(int i = 1; i <= n; i++){
8         cin >> a[i];
9         x[i] = INT_MAX;
10    }
11    f[0]=0;
12    int ans=0;
13    for(int i = 1; i <= n; i++){
14        int l = 0, r = i;
15        while (l+1<r){
16            int m=(l+r)/2;
17            if (x[m]<a[i]){
18                l=m;
19            }else{
20                r=m;
21            }
22            // change to x[m]<=a[i] for
            // non-decreasing case
23        }
24        f[i]=l+1;
25        x[l+1]=a[i];
26        if(f[i]>ans){
27            ans=f[i];
28        }
29    }
30    cout << ans << endl;
31    return 0;
32 }

```

## 2.6 Longest Palindromic Subsequence(LPS)

```

1 int LPS(vector<vector<int>>& dp, string& s, int l,
2 int r){
3     if(-dp[l][r]) return dp[l][r];
4     if(l == r) return dp[l][r] = 1;
5     if(l > r) return dp[l][r] = 0;
6     if(s[l] == s[r]) return dp[l][r] = LPS(dp, s,
7         l+1, r-1) + 2;
8     return dp[l][r] = max(LPS(dp, s, l + 1, r),
9         LPS(dp, s, l, r - 1));
10 }

```

## 3 Graph Theory

### 3.1 Lowest Common Ancestor(LCA)

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int LOG = 20;
4 int par[N][LOG];
5 int tin[N], tout[N];
6 int timer = 0;
7 void dfs(int v, int p){
8     tin[v] = ++timer;
9     par[v][0] = p;
10    for (int it : G[v]){
11        if (it != p){
12            dfs(it, v);
13        }
14    }
15    tout[v] = ++timer;
16 }
17 void Doubling(){
18     for (int i = 1; i < N; ++i){
19         for (int j = 1; j < LOG; ++j){
20             par[i][j] = par[par[i][j-1]][j-1];
21         }
22     }
23 }
24 bool anc(int v, int u){
25     return tin[v] <= tin[u] && tout[u] <= tout[v];
26 }
27 int LCA(int v, int u){
28     if (anc(v, u)){
29         return v;
30     }
31     for (int j = LOG - 1; j >= 0; --j){
32         if (!anc(par[v][j], u)){
33             v = par[v][j];
34         }
35     }
36     return par[v][0];
37 }

```

### 3.2 bellman-ford

```

1 void bellman(vector<edge>& edges, vector<int>& dist,
2 int n){
3     // n - 1 relax
4     for(int i = 1; i < n; i++){
5         for(edge e : edges){
6             int u = e.start, v = e.end, d = e.dist;
7             if(dist[u] == INF) continue;
8             dist[v] = min(dist[v], d + dist[u]);
9         }
10    }
11    // 偵測負權迴路
12    for(edge e : edges){
13        int u = e.start, v = e.end, d = e.dist;
14        if(dist[u] == INF) continue;
15        if(dist[v] > dist[v] + d){
16            printf("It is contained negative
17            cycle.\n");
18            break;
19        }
20    }
21 }

```

### 3.3 dijkstra

```

1 void dijkstra(ll st){
2     vector<ll> dis(n, INF);
3     //p11 (vetrex, distance)

```

```

4 priority_queue<pll, vector<pll>, greater<pll>> pq;
5 pq.push({st, 0});
6 dis[st] = 0;
7 while(!pq.empty()){
8     pll now = pq.top();
9     pq.pop();
10    if(now.ss != dis[now.ff]) continue;
11    for(pll i : graph[t.ff]){
12        if(dis[now.ff] + i.ss < dis[i.ff]){
13            dis[i.ff] = now.ss + i.ss;
14            pq.push({i.ff, dis[i.ff]});
15        }
16    }
17 }
18 }

```

### 3.4 Topological

```

1 //此處為建立 Adjacency List 和每個節點的入度點數量
2 vector<vector<int>> make(vector<Edge>& nodes, int n){
3     vector<vector<int>> graph(n + 1);
4     vector<int> indegree(n, 0);
5     for(auto node : nodes){
6         graph[node.src].pb(node.des);
7         indegree[node.des]++; //計算入度點
8     }
9     graph[n] = indegree;
10    return graph;
11 }
12
13 // 拓模排序
14 vector<int> TopologicalOrder(vector<vector<int>>
15     graph){
16     int n = graph.size();
17     queue<int> q;
18     vector<int> result;
19     for(int i=0; i<graph[n-1].size(); i++){
20         if(!graph[n-1][i]) q.push(i);
21     }
22     while(!q.empty()){
23         int cnt = q.front();
24         result.pb(cnt);
25         q.pop();
26         for(int i=0; i<graph[cnt].size(); i++){
27             graph[n-1][graph[cnt][i]]--;
28             if(!graph[n-1][graph[cnt][i]])
29                 q.push(graph[cnt][i]);
30         }
31     }
32     //偵測循環
33     for(auto i : graph[n-1]) if(i) return {};
34     return result;
35 }

```

## 4 Algorithm

### 4.1 Ternary Search

```

1 int l = -10000;
2 int r = 10000;
3 int iterations = 100;
4 for (int i = 0; i < iterations; i++){
5     double mr = (l + r) / 2.0;
6     double ml = (l + mr) / 2.0;
7     // f( ): 目標函數
8     if (f(ml) < f(mr)) r = mr;
9     else l = ml;
10 }

```

## 5 Number Theory

### 5.1 質數篩法 Sieve of Eratosthenes

```

1 bool a[46342];
2 vector <int> v;
3 for (int j = 2; j < 46342; j++){
4     if (!a[j]){
5         v.push_back(j);
6         for (int i = j * j; i < 46342; i += j){
7             a[i] = true;
8         }
9     }
10 }

```

## 6 Data Structure

### 6.1 Disjoint Set Union-Find

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 vector<int> dsu, rk;
5
6 void initDSU(int n){
7     dsu.resize(n);
8     rk.resize(n);
9     for(int i = 0; i < n; i++) dsu[i] = i, rk[i] = 1;
10 }
11
12 int findDSU(int x){
13     if(dsu[x] == x) return x;
14     dsu[x] = findDSU(dsu[x]);
15     return dsu[x];
16 }
17
18 void unionDSU(int a, int b){
19     int pa = findDSU(a), pb = findDSU(b);
20     if(rk[pa] > rk[pb]) swap(pa, pb);
21     if(rk[pa] == rk[pb]) rk[pb]++;
22     dsu[pa] = pb;
23 }

```

### 6.2 Segment Tree

```

1 #include <bits/stdc++.h>
2 #define ll long long
3 using namespace std;
4
5 struct segtree {
6
7     vector<ll> sums;
8     ll size;
9
10    // 線段樹初始化
11    void init(ll n){
12        size = 1;
13        while(size < n) size<<1;
14        sums.assign(size<<1, 0LL);
15    }
16
17    // 更新數值
18    void update(ll i, ll v, ll x, ll Lptr, ll Rptr){
19        if(Rptr - Lptr == 1){
20            sums[x] = v;
21            return;
22        }
23        ll m = (Lptr + Rptr) / 2;
24        if(i < m) update(i, v, 2*x+1, Lptr, m);
25        else update(i, v, 2*x+2, m, Rptr);

```

```

26     sums[x] = sums[2*x+1] + sums[2*x+2];
27 }
28
29 void update(ll a, ll b){
30     update(a, b, 0, 0, size);
31 }
32
33 // 查詢資訊
34 ll query(ll l, ll r, ll x, ll Lptr, ll Rptr){
35     if( Lptr >= r || Rptr <= l ) return 0;
36     if( Lptr >= l && Rptr <= r ) return sums[x];
37     ll m = (Lptr + Rptr) / 2;
38     ll s1 = query(l, r, 2*x+1, Lptr, m);
39     ll s2 = query(l, r, 2*x+2, m, Rptr);
40     return s1 + s2;
41 }
42
43 ll query(ll a, ll b){
44     return query(a, b, 0, 0, size);
45 }
46 };

```

## 7 String

### 7.1 Suffix Array

```

1 #include<bits/stdc++.h>
2 #define int long long
3
4 using namespace std;
5
6 void count_sort(auto &p, auto &c){
7     int n = p.size();
8     vector<int> cnt(n);
9     for(auto el : c) cnt[el] ++;
10    vector<int> p_new(n), pos(n);
11    pos[0] = 0;
12    for(int i=1;i<n;i++) pos[i] = pos[i-1] + cnt[i-1];
13    for(auto el : p){
14        int i = c[el];
15        p_new[pos[i]] = el;
16        pos[i] ++;
17    }
18    p = p_new;
19 }
20
21 signed main(){
22     string s;
23     cin>>s;
24     s += "$";
25     int n = s.size();
26     vector<pair<char, int>> v(n);
27     vector<int> p(n), c(n);
28     for(int i=0;i<n;i++) v[i] = {s[i], i};
29     sort(v.begin(), v.end());
30
31     for(int i=0;i<v.size();i++) p[i] = v[i].second;
32     c[p[0]] = 0;
33     for(int i=1;i<v.size();i++){
34         if(v[i].first == v[i-1].first) c[p[i]] =
35             c[p[i-1]];
36         else c[p[i]] = c[p[i-1]] + 1;
37     }
38
39     int k = 0;
40     while((1 << k) < n){
41         for(int i=0;i<n;i++) p[i] = (p[i] - (1 << k) + n)
42             % n;
43         count_sort(p, c);
44
45         vector<int> c_new(n);
46         c_new[p[0]] = 0;
47         for(int i=1;i<v.size();i++){

```

```

46     pair<int, int> prev = {c[p[i-1]], c[(p[i-1] +
47         (1 << k)) % n]};
48     pair<int, int> now = {c[p[i]], c[(p[i] + (1 <<
49         k)) % n]};
50     if(prev == now) c_new[p[i]] = c_new[p[i-1]];
51     else c_new[p[i]] = c_new[p[i-1]] + 1;
52 }
53 c = c_new;
54 k++;
55 }
56 for(int i=0;i<n;i++) cout<<p[i]<<"\n";
57 }

```

### 7.2 Suffix Array LCP

```

1 #include <bits/stdc++.h>
2 #define int long long
3 using namespace std;
4
5 vector<int> lcp(n);
6 int k = 0;
7 for(int i=0;i<n-1;i++){
8     int pi = c[i];
9     int j = p[pi - 1];
10    while(s[i+k] == s[j+k]) k++;
11    lcp[pi] = k;
12    k = k-1 > 0 ? k-1 : 0;
13 }

```

### 7.3 KMP algorithm

```

1 vector<int> NEXT;
2 void getNext(string p){
3     int i = 1, j = i - 1;
4     while(i < p.size()){
5         if(p[i] == p[j]){
6             NEXT[i++] = ++j;
7         }
8         else if(j <= 0){
9             NEXT[i++] = 0;
10        }
11        else{
12            j = NEXT[j - 1];
13        }
14    }
15 }
16
17 int KMP(string s, string p){
18     int i = 0, j = 0;
19     while(i < s.size() && j < p.size()){
20         if(s[i] == p[j]){
21             ++i;
22             ++j;
23         }else if(j <= 0){
24             ++i;
25         }else{
26             j = NEXT[j - 1];
27         }
28     }
29
30     if(j >= p.size()) return i - p.size();
31     else return -1;
32 }

```

### 7.4 Manacher's algorithm

```

1 int P[SIZE * 2];
2
3 string convertToNewString(const string &s) {
4     string newString = "@";
5

```

```

6   for (int i = 0; i < s.size(); i++) {
7       newString += "#" + s.substr(i, 1);
8   }
9
10  newString += "$";
11  return newString;
12 }
13
14 string longestPalindromeSubstring(const string &s) {
15     string Q = convertToNewString(s);
16     int c = 0, r = 0;                // current
17                                     // center, right limit
18
19     for (int i = 1; i < Q.size() - 1; i++) {
20         // find the corresponding letter in the
21         // palidrome subString
22         int iMirror = c - (i - c);
23
24         if (r > i) {
25             P[i] = min(r - i, P[iMirror]);
26         }
27
28         // expanding around center i
29         while (Q[i + 1 + P[i]] == Q[i - 1 - P[i]]){
30             P[i]++;
31         }
32
33         // Update c,r in case if the palindrome
34         // centered at i expands past r,
35         if (i + P[i] > r) {
36             c = i;                // next center = i
37             r = i + P[i];
38         }
39     }
40
41     // Find the longest palindrome length in p.
42
43     int maxPalindrome = 0;
44     int centerIndex = 0;
45
46     for (int i = 1; i < Q.size() - 1; i++) {
47         if (P[i] > maxPalindrome) {
48             maxPalindrome = P[i];
49             centerIndex = i;
50         }
51     }
52
53     cout << maxPalindrome << "\n";
54     return s.substr( (centerIndex - 1 -
55                     maxPalindrome) / 2, maxPalindrome);
56 }

```

```

3
4 int main()
5 {
6     vector<int> a = {1561, 777, 89898, 5}; // --> {3,
7                                     // 2, 4, 1}
8     vector<int> b = a;
9
10    sort(b.begin(), b.end());
11    b.resize(unique(b.begin(), b.end()) - b.begin());
12
13    for(int i:a)
14    {
15        cout << lower_bound(b.begin(), b.end(), i) -
16            b.begin() + 1 << "\n";
17    }
18    return 0;
19 }

```

## 8.2 Map + Set ( $O(N \log N)$ )

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     vector<int> a = {1561, 777, 89898, 5}; // -> {3,
7                                     // 2, 4, 1}
8
9     int now = 1;
10    map<int, int> mp;
11    set<int> ms;
12
13    for(int i:a)
14    {
15        ms.insert(i);
16    }
17
18    for(int i:ms)
19    {
20        mp[i] = now++;
21    }
22
23    for(int i:a)
24    {
25        cout << mp[i] << "\n";
26    }
27
28    return 0;
29 }

```

## 7.5 Z-algorithm

```

1 void z_build(const char *S, int *Z) {
2     Z[0] = 0;
3     int bst = 0;
4     for(int i = 1; S[i]; i++) {
5         if(Z[bst] + bst < i) Z[i] = 0;
6         else Z[i] = min(Z[bst]+bst-i, Z[i-bst]);
7         while(S[Z[i]] == S[i+Z[i]]) Z[i]++;
8         if(Z[i] + i > Z[bst] + bst) bst = i;
9     }
10 }

```

# 8 離散化 Discretization

## 8.1 Vector ( $O(N \log N)$ )

```

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

```