

DS期末机考模板20230218

0. 代码模板

```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include<bits/stdc++.h>
3  using namespace std;
4  typedef long long ll;
5  #define CaseT int CaseT; cin >> CaseT; while(CaseT--)
6  #define endl '\n'
7  #define all(x) (x).begin(), (x).end()
8  #define rall(x) (x).rbegin(), (x).rend()
9  const int INF = 0x3f3f3f3f;
10
11 void solve() {
12
13 }
14
15 int main() {
16     cin.tie(0)->sync_with_stdio(false);
17 #ifndef ONLINE_JUDGE
18     freopen("in.txt", "r", stdin);
19     freopen("out.txt", "w", stdout);
20 #endif
21     // init();
22     // CaseT
23     solve();
24     return 0;
25 }

```

1. 二叉搜索树

```

1  const int MAXN = 1e3 + 5;
2  struct BinarySearchTree {
3      struct Node {
4          int l, r;
5          int val;
6          int cnt;
7      }BST[MAXN];
8      int root;
9      int idx;
10
11     BinarySearchTree() {
12         build();
13     }
14
15     int newNode(int x) {
16         BST[++idx].val = x;
17         BST[idx].cnt = 1;
18         BST[idx].l = BST[idx].r = 0;
19         return idx;

```

```

20 }
21
22 void build() {
23     root = 1, idx = 0;
24     newNode(-INF), newNode(INF);
25     BST[root].r = 2;
26 }
27
28 void inorderTraversal(int u, vector<int>& res) {
29     if (BST[u].l) inorderTraversal(BST[u].l, res);
30     if (abs(BST[u].val) != INF)
31         for (int i = 0; i < BST[u].cnt; i++) res.push_back(BST[u].val);
32     if (BST[u].r) inorderTraversal(BST[u].r, res);
33 }
34
35 void print() {
36     vector<int> res;
37     inorderTraversal(root, res);
38     for (auto u : res) cout << u << ' ';
39     cout << endl;
40 }
41
42 void insert(int& u, int x) {
43     if (!u) u = newNode(x);
44     else if (BST[u].val == x) BST[u].cnt++;
45     else if (BST[u].val > x) insert(BST[u].l, x);
46     else insert(BST[u].r, x);
47 }
48
49 int getPrev(int u, int x) {
50     if (!u) return -INF;
51     else if (BST[u].val < x) return u;
52     else if (BST[u].val >= x) return getPrev(BST[u].l, x);
53     else return getPrev(BST[u].r, x);
54 }
55
56 void remove(int& u, int x) {
57     if (!u) return;
58
59     if (BST[u].val == x) {
60         if (BST[u].cnt > 1) BST[u].cnt--;
61         else if (!BST[u].l && !BST[u].r) u = 0;
62         else if (BST[u].l && !BST[u].r) u = BST[u].l;
63         else if (!BST[u].l && BST[u].r) u = BST[u].r;
64         else {
65             int pre = getPrev(u, x);
66             int r = BST[u].r;
67             BST[u = pre].r = r;
68         }
69     }
70     else if (BST[u].val > x) remove(BST[u].l, x);
71     else remove(BST[u].r, x);
72 }
73
74 int find(int u, int x, int step) {
75     if (!u) return INF;
76     else if (BST[u].val == x) return step;
77     else if (BST[u].val > x) return find(BST[u].l, x, step + 1);

```

```

78     else return find(BST[u].r, x, step + 1);
79 }
80 };

```

2. 根据树的遍历建二叉树

2.1 根据先序遍历建二叉树

给定二叉树的先序遍历,求其高度.

```

1  const int MAXN = 1e3 + 5;
2  struct BinaryTree {
3      int n; // s的长度
4      string s; // 先序遍历
5      int begin;
6
7      struct Node {
8          char data;
9          int l, r;
10     }Tree[MAXN];
11     int idx;
12
13     BinaryTree(string _s) :s(_s) {
14         n = s.length();
15         begin = idx = 0;
16         build();
17     }
18
19     int build() {
20         if (begin == n) return 0;
21
22         if (s[begin] == '0') {
23             begin++;
24             return 0;
25         }
26
27         int cur = ++idx;
28         Tree[cur].data = s[begin++];
29         Tree[cur].l = build(), Tree[cur].r = build();
30         return cur;
31     }
32
33     void print() {
34         for (int i = 1; i <= n; i++) {
35             cout << i << ' ' << Tree[i].data << ' '
36                  << Tree[i].l << ' ' << Tree[i].r << endl;
37         }
38     }
39
40     int getHeight(int u) {
41         if (!u) return 0;
42         else return max(getHeight(Tree[u].l), getHeight(Tree[u].r)) + 1;
43     }
44 };
45

```

```

46 void solve() {
47     string s; cin >> s;
48
49     BinaryTree tr(s);
50     cout << tr.getHeight(1) << endl;
51 }

```

2.2 先序+中序还原二叉树

```

1  const int MAXN = 1e3 + 5;
2  struct BinaryTree {
3      struct Node {
4          char data;
5          int l, r;
6      }Tree[MAXN];
7      int idx;
8
9      BinaryTree(int _n, string _pre, string _in) {
10         idx = 0;
11         build(_pre, _in, _n);
12     }
13
14     int build(string pre, string in, int n) { // 要建立的节点数
15         if (!n) return 0;
16
17         int cur = ++idx;
18         Tree[cur].data = pre[0];
19
20         int pos = in.find(pre[0]);
21         Tree[cur].l = build(pre.substr(1), in, pos);
22         Tree[cur].r = build(pre.substr(pos + 1), in.substr(pos + 1), n - pos - 1);
23         return cur;
24     }
25
26     int getHeight(int u) {
27         if (!u) return 0;
28         else return max(getHeight(Tree[u].l), getHeight(Tree[u].r)) + 1;
29     }
30 };
31
32 void solve() {
33     int n; string pre, in; cin >> n >> pre >> in;
34
35     BinaryTree tr(n, pre, in);
36     cout << tr.getHeight(1) << endl;
37 }

```

2.3 后序+中序还原二叉树

```

1  const int MAXN = 1e3 + 5;
2  struct BinaryTree {
3      struct Node {
4          char data;
5          int l, r;

```

```

6   }Tree[MAXN];
7   int idx;
8
9   BinaryTree(int _n, string _post, string _in) {
10      idx = 0;
11      build(_post, _in, _n);
12  }
13
14  int build(string post, string in, int n) { // 要建立的节点数
15      if (!n) return 0;
16
17      int cur = ++idx;
18      Tree[cur].data = post.back();
19
20      int pos = in.find(post.back());
21      Tree[cur].r = build(post.substr(pos, n - pos - 1),
22                          in.substr(pos + 1), n - pos - 1);
23      Tree[cur].l = build(post.substr(0, pos), in, pos);
24      return cur;
25  }
26
27  void inorderTraversal(int u, vector<char>& res) {
28      res.push_back(Tree[u].data);
29      if (Tree[u].l) inorderTraversal(Tree[u].l, res);
30      if (Tree[u].r) inorderTraversal(Tree[u].r, res);
31  }
32 };
33
34 void solve() {
35     int n; cin >> n;
36     string post, in;
37     for (int i = 0; i < n; i++) {
38         int ch; cin >> ch;
39         post.push_back(ch);
40     }
41     for (int i = 0; i < n; i++) {
42         int ch; cin >> ch;
43         in.push_back(ch);
44     }
45
46     BinaryTree tr(n, post, in);
47     vector<char> res;
48     tr.inorderTraversal(1, res);
49     cout << "Preorder: ";
50     for (auto u : res) cout << (int)u << " \n"[u == res.back()];
51 }

```

3. 二叉树的遍历

3.1 递归实现

```

1  const int MAXN = 1e3 + 5;
2  struct BinaryTree {
3      int n; // s的长度
4      string s; // 先序遍历
5      int begin;
6
7      struct Node {
8          char data;
9          int l, r;
10     }Tree[MAXN];
11     int idx;
12
13     BinaryTree(string _s) :s(_s) {
14         n = s.length();
15         begin = idx = 0;
16         build();
17     }
18
19     int build() {
20         if (begin == n) return 0;
21
22         if (s[begin] == '#') {
23             begin++;
24             return 0;
25         }
26
27         int cur = ++idx;
28         Tree[cur].data = s[begin++];
29         Tree[cur].l = build(), Tree[cur].r = build();
30         return cur;
31     }
32
33     void print() {
34         for (int i = 1; i <= n; i++) {
35             cout << i << ' ' << Tree[i].data << ' '
36                 << Tree[i].l << ' ' << Tree[i].r << endl;
37         }
38     }
39
40     void bfs(vector<char>& res) {
41         queue<int> que;
42         que.push(1);
43
44         while (que.size()) {
45             auto u = que.front(); que.pop();
46             res.push_back(Tree[u].data);
47
48             if (Tree[u].l) que.push(Tree[u].l);
49             if (Tree[u].r) que.push(Tree[u].r);
50         }
51     }
52
53     void preorderTraversal(int u, vector<char>& res) {
54         res.push_back(Tree[u].data);
55         if (Tree[u].l) preorderTraversal(Tree[u].l, res);

```

```

56     if (Tree[u].r) preorderTraversal(Tree[u].r, res);
57 }
58
59 void inorderTraversal(int u, vector<char>& res) {
60     if (Tree[u].l) inorderTraversal(Tree[u].l, res);
61     res.push_back(Tree[u].data);
62     if (Tree[u].r) inorderTraversal(Tree[u].r, res);
63 }
64
65 void postorderTraversal(int u, vector<char>& res) {
66     if (Tree[u].l) postorderTraversal(Tree[u].l, res);
67     if (Tree[u].r) postorderTraversal(Tree[u].r, res);
68     res.push_back(Tree[u].data);
69 }
70 };

```

3.2 非递归实现

```

1  const int MAXN = 1e3 + 5;
2  struct BinaryTree {
3      int n; // s的长度
4      string s; // 先序遍历
5      int begin;
6
7      struct Node {
8          char data;
9          int l, r;
10     }Tree[MAXN];
11     int idx;
12
13     BinaryTree(string _s) :s(_s) {
14         n = s.length();
15         begin = idx = 0;
16         build();
17     }
18
19     int build() {
20         if (begin == n) return 0;
21
22         if (s[begin] == '#') {
23             begin++;
24             return 0;
25         }
26
27         int cur = ++idx;
28         Tree[cur].data = s[begin++];
29         Tree[cur].l = build(), Tree[cur].r = build();
30         return cur;
31     }
32
33     void print() {
34         for (int i = 1; i <= n; i++) {
35             cout << i << ' ' << Tree[i].data << ' '
36                 << Tree[i].l << ' ' << Tree[i].r << endl;
37         }
38     }

```

```

39
40 void preorderTraversal(int u, vector<char>& res) {
41     stack<int> stk;
42     stk.push(u);
43
44     while (stk.size()) {
45         auto v = stk.top(); stk.pop();
46         res.push_back(Tree[v].data);
47
48         if (Tree[v].r) stk.push(Tree[v].r);
49         if (Tree[v].l) stk.push(Tree[v].l);
50     }
51 }
52
53 void inorderTraversal(int u, vector<char>& res) {
54     stack<int> stk;
55     while (u || stk.size()) {
56         while (u) {
57             stk.push(u);
58             u = Tree[u].l;
59         }
60
61         u = stk.top(); stk.pop();
62         res.push_back(Tree[u].data);
63         u = Tree[u].r;
64     }
65 }
66
67 void postorderTraversal(int u, vector<char>& res) {
68     stack<int> stk;
69     int last = -1;
70
71     while (u || stk.size()) {
72         while (u) {
73             stk.push(u);
74             u = Tree[u].l;
75         }
76
77         u = stk.top();
78         if (!Tree[u].r || last == Tree[u].r) {
79             res.push_back(Tree[u].data);
80             stk.pop();
81             last = u;
82             u = 0;
83         }
84         else u = Tree[u].r;
85     }
86 }
87 };
88
89 void solve() {
90     string s; cin >> s;
91     BinaryTree tr(s);
92
93     cout << "Inorder: ";
94     vector<char> res;
95     tr.inorderTraversal(1, res);
96     for (auto u : res) cout << u << " \n"[u == res.back()];

```



```

97
98     cout << "Preorder: ";
99     res.clear();
100    tr.preorderTraversal(1, res);
101    for (auto u : res) cout << u << " \n"[u == res.back()];
102
103    cout << "Postorder: ";
104    res.clear();
105    tr.postorderTraversal(1, res);
106    for (auto u : res) cout << u << " \n"[u == res.back()];
107 }

```

4. KMP

4.1 KMP模板

```

1  struct KMP {
2      int n, m; // 模式串、文本串长度
3      string p, s; // 模式串、文本串
4      vector<int> nxt;
5
6      KMP(string _p, string _s = "") :
7      p(_p), n(_p.length()), s(_s), m(_s.length()) {
8          nxt.resize(n + 1);
9
10         for (int i = 2, j = 0; i <= n; i++) {
11             while (j && p[i] != p[j + 1]) j = nxt[j];
12             if (p[i] == p[j + 1]) j++;
13             nxt[i] = j;
14         }
15     }
16
17     vector<int> kmp() {
18         vector<int> res;
19         for (int i = 1, j = 0; i <= m; i++) {
20             while (j && s[i] != p[j + 1]) j = nxt[j];
21             if (s[i] == p[j + 1]) j++;
22
23             if (j == n) {
24                 res.push_back(i - n);
25                 j = nxt[j];
26             }
27         }
28         return res;
29     }
30 };
31
32 void solve() {
33     string s, p; cin >> s >> p;
34
35     KMP solver(p, s);
36
37     int len = (int)p.length();
38     cout << -1 << ' ';
39     for (int i = 1; i < len; i++) cout << solver.nxt[i] << ' ';

```

```

40     cout << endl;
41
42     auto ans = solver.kmp();
43     cout << (ans.size() ? ans[0] + 1 : 0) << endl;
44 }

```

4.2 求字符串的最长不重叠公共前后缀

```

1 void solve() {
2     string p; cin >> p;
3
4     KMP solver(p);
5     int len = p.length();
6     if (!solver.nxt[len]) cout << "empty" << endl;
7     else {
8         while (solver.nxt[len] * 2 > len) len = solver.nxt[len];
9         cout << p.substr(0, solver.nxt[len]) << endl;
10    }
11 }

```

4.3 求字符串周期

给定一个字符串,求至少需在末尾添加多少个字符可使得该串由某个不为自身的字串循环而成.

```

1 void solve() {
2     string p; cin >> p;
3
4     KMP solver(p);
5     int len = p.length(), t = len - solver.nxt[len];
6     cout << (t == len ? len : (len + t - 1) / t * t - len) << endl;
7 }

```

5. Huffman树与Huffman编码

```

1 const int MAXN = 1e3 + 5;
2 struct HuffmanTree {
3     int n; // 原序列长度
4
5     struct Node {
6         int id;
7         char data;
8         int w;
9         int l, r;
10        int fa;
11
12        Node() : l(0), r(0), fa(0) {}
13        Node(int _id, int _w, int _l, int _r) : id(_id), w(_w), l(_l), r(_r) {}
14
15        bool operator>(const Node& p) const {
16            return w != p.w ? w > p.w : id > p.id;
17        }
18    };
19
20    Node* root;
21    Node* nodes[MAXN];
22
23    void init() {
24        root = new Node(1, 1, 0, 0);
25        for (int i = 2; i <= n; i++) {
26            nodes[i] = new Node(i, 1, 0, 0);
27        }
28    }
29
30    void build() {
31        while (nodes[1].l || nodes[1].r) {
32            int l = nodes[1].l, r = nodes[1].r;
33            nodes[l].fa = nodes[r].fa = 1;
34            nodes[1].l = nodes[l].id, nodes[1].r = nodes[r].id;
35            nodes[1].w = nodes[l].w + nodes[r].w;
36        }
37    }
38
39    void dfs(int id, int w) {
40        if (!id) return;
41        cout << nodes[id].data << " ";
42        if (w > 1) dfs(id, w - 1);
43    }
44
45    void print() {
46        dfs(1, n);
47        cout << endl;
48    }
49
50    void solve() {
51        init();
52        build();
53        print();
54    }
55
56    void solve2() {
57        init();
58        build();
59        print();
60    }
61
62    void solve3() {
63        init();
64        build();
65        print();
66    }
67
68    void solve4() {
69        init();
70        build();
71        print();
72    }
73
74    void solve5() {
75        init();
76        build();
77        print();
78    }
79
80    void solve6() {
81        init();
82        build();
83        print();
84    }
85
86    void solve7() {
87        init();
88        build();
89        print();
90    }
91
92    void solve8() {
93        init();
94        build();
95        print();
96    }
97
98    void solve9() {
99        init();
100       build();
101       print();
102   }
103
104   void solve10() {
105       init();
106       build();
107       print();
108   }
109
110   void solve11() {
111       init();
112       build();
113       print();
114   }
115
116   void solve12() {
117       init();
118       build();
119       print();
120   }
121
122   void solve13() {
123       init();
124       build();
125       print();
126   }
127
128   void solve14() {
129       init();
130       build();
131       print();
132   }
133
134   void solve15() {
135       init();
136       build();
137       print();
138   }
139
140   void solve16() {
141       init();
142       build();
143       print();
144   }
145
146   void solve17() {
147       init();
148       build();
149       print();
150   }
151
152   void solve18() {
153       init();
154       build();
155       print();
156   }
157
158   void solve19() {
159       init();
160       build();
161       print();
162   }
163
164   void solve20() {
165       init();
166       build();
167       print();
168   }
169
170   void solve21() {
171       init();
172       build();
173       print();
174   }
175
176   void solve22() {
177       init();
178       build();
179       print();
180   }
181
182   void solve23() {
183       init();
184       build();
185       print();
186   }
187
188   void solve24() {
189       init();
190       build();
191       print();
192   }
193
194   void solve25() {
195       init();
196       build();
197       print();
198   }
199
200   void solve26() {
201       init();
202       build();
203       print();
204   }
205
206   void solve27() {
207       init();
208       build();
209       print();
210   }
211
212   void solve28() {
213       init();
214       build();
215       print();
216   }
217
218   void solve29() {
219       init();
220       build();
221       print();
222   }
223
224   void solve30() {
225       init();
226       build();
227       print();
228   }
229
230   void solve31() {
231       init();
232       build();
233       print();
234   }
235
236   void solve32() {
237       init();
238       build();
239       print();
240   }
241
242   void solve33() {
243       init();
244       build();
245       print();
246   }
247
248   void solve34() {
249       init();
250       build();
251       print();
252   }
253
254   void solve35() {
255       init();
256       build();
257       print();
258   }
259
260   void solve36() {
261       init();
262       build();
263       print();
264   }
265
266   void solve37() {
267       init();
268       build();
269       print();
270   }
271
272   void solve38() {
273       init();
274       build();
275       print();
276   }
277
278   void solve39() {
279       init();
280       build();
281       print();
282   }
283
284   void solve40() {
285       init();
286       build();
287       print();
288   }
289
290   void solve41() {
291       init();
292       build();
293       print();
294   }
295
296   void solve42() {
297       init();
298       build();
299       print();
300   }
301
302   void solve43() {
303       init();
304       build();
305       print();
306   }
307
308   void solve44() {
309       init();
310       build();
311       print();
312   }
313
314   void solve45() {
315       init();
316       build();
317       print();
318   }
319
320   void solve46() {
321       init();
322       build();
323       print();
324   }
325
326   void solve47() {
327       init();
328       build();
329       print();
330   }
331
332   void solve48() {
333       init();
334       build();
335       print();
336   }
337
338   void solve49() {
339       init();
340       build();
341       print();
342   }
343
344   void solve50() {
345       init();
346       build();
347       print();
348   }
349
350   void solve51() {
351       init();
352       build();
353       print();
354   }
355
356   void solve52() {
357       init();
358       build();
359       print();
360   }
361
362   void solve53() {
363       init();
364       build();
365       print();
366   }
367
368   void solve54() {
369       init();
370       build();
371       print();
372   }
373
374   void solve55() {
375       init();
376       build();
377       print();
378   }
379
380   void solve56() {
381       init();
382       build();
383       print();
384   }
385
386   void solve57() {
387       init();
388       build();
389       print();
390   }
391
392   void solve58() {
393       init();
394       build();
395       print();
396   }
397
398   void solve59() {
399       init();
400       build();
401       print();
402   }
403
404   void solve60() {
405       init();
406       build();
407       print();
408   }
409
410   void solve61() {
411       init();
412       build();
413       print();
414   }
415
416   void solve62() {
417       init();
418       build();
419       print();
420   }
421
422   void solve63() {
423       init();
424       build();
425       print();
426   }
427
428   void solve64() {
429       init();
430       build();
431       print();
432   }
433
434   void solve65() {
435       init();
436       build();
437       print();
438   }
439
440   void solve66() {
441       init();
442       build();
443       print();
444   }
445
446   void solve67() {
447       init();
448       build();
449       print();
450   }
451
452   void solve68() {
453       init();
454       build();
455       print();
456   }
457
458   void solve69() {
459       init();
460       build();
461       print();
462   }
463
464   void solve70() {
465       init();
466       build();
467       print();
468   }
469
470   void solve71() {
471       init();
472       build();
473       print();
474   }
475
476   void solve72() {
477       init();
478       build();
479       print();
480   }
481
482   void solve73() {
483       init();
484       build();
485       print();
486   }
487
488   void solve74() {
489       init();
490       build();
491       print();
492   }
493
494   void solve75() {
495       init();
496       build();
497       print();
498   }
499
500   void solve76() {
501       init();
502       build();
503       print();
504   }
505
506   void solve77() {
507       init();
508       build();
509       print();
510   }
511
512   void solve78() {
513       init();
514       build();
515       print();
516   }
517
518   void solve79() {
519       init();
520       build();
521       print();
522   }
523
524   void solve80() {
525       init();
526       build();
527       print();
528   }
529
530   void solve81() {
531       init();
532       build();
533       print();
534   }
535
536   void solve82() {
537       init();
538       build();
539       print();
540   }
541
542   void solve83() {
543       init();
544       build();
545       print();
546   }
547
548   void solve84() {
549       init();
550       build();
551       print();
552   }
553
554   void solve85() {
555       init();
556       build();
557       print();
558   }
559
560   void solve86() {
561       init();
562       build();
563       print();
564   }
565
566   void solve87() {
567       init();
568       build();
569       print();
570   }
571
572   void solve88() {
573       init();
574       build();
575       print();
576   }
577
578   void solve89() {
579       init();
580       build();
581       print();
582   }
583
584   void solve90() {
585       init();
586       build();
587       print();
588   }
589
590   void solve91() {
591       init();
592       build();
593       print();
594   }
595
596   void solve92() {
597       init();
598       build();
599       print();
600   }
601
602   void solve93() {
603       init();
604       build();
605       print();
606   }
607
608   void solve94() {
609       init();
610       build();
611       print();
612   }
613
614   void solve95() {
615       init();
616       build();
617       print();
618   }
619
620   void solve96() {
621       init();
622       build();
623       print();
624   }
625
626   void solve97() {
627       init();
628       build();
629       print();
630   }
631
632   void solve98() {
633       init();
634       build();
635       print();
636   }
637
638   void solve99() {
639       init();
640       build();
641       print();
642   }
643
644   void solve100() {
645       init();
646       build();
647       print();
648   }
649
650   void solve101() {
651       init();
652       build();
653       print();
654   }
655
656   void solve102() {
657       init();
658       build();
659       print();
660   }
661
662   void solve103() {
663       init();
664       build();
665       print();
666   }
667
668   void solve104() {
669       init();
670       build();
671       print();
672   }
673
674   void solve105() {
675       init();
676       build();
677       print();
678   }
679
680   void solve106() {
681       init();
682       build();
683       print();
684   }
685
686   void solve107() {
687       init();
688       build();
689       print();
690   }
691
692   void solve108() {
693       init();
694       build();
695       print();
696   }
697
698   void solve109() {
699       init();
700       build();
701       print();
702   }
703
704   void solve110() {
705       init();
706       build();
707       print();
708   }
709
710   void solve111() {
711       init();
712       build();
713       print();
714   }
715
716   void solve112() {
717       init();
718       build();
719       print();
720   }
721
722   void solve113() {
723       init();
724       build();
725       print();
726   }
727
728   void solve114() {
729       init();
730       build();
731       print();
732   }
733
734   void solve115() {
735       init();
736       build();
737       print();
738   }
739
740   void solve116() {
741       init();
742       build();
743       print();
744   }
745
746   void solve117() {
747       init();
748       build();
749       print();
750   }
751
752   void solve118() {
753       init();
754       build();
755       print();
756   }
757
758   void solve119() {
759       init();
760       build();
761       print();
762   }
763
764   void solve120() {
765       init();
766       build();
767       print();
768   }
769
770   void solve121() {
771       init();
772       build();
773       print();
774   }
775
776   void solve122() {
777       init();
778       build();
779       print();
780   }
781
782   void solve123() {
783       init();
784       build();
785       print();
786   }
787
788   void solve124() {
789       init();
790       build();
791       print();
792   }
793
794   void solve125() {
795       init();
796       build();
797       print();
798   }
799
800   void solve126() {
801       init();
802       build();
803       print();
804   }
805
806   void solve127() {
807       init();
808       build();
809       print();
810   }
811
812   void solve128() {
813       init();
814       build();
815       print();
816   }
817
818   void solve129() {
819       init();
820       build();
821       print();
822   }
823
824   void solve130() {
825       init();
826       build();
827       print();
828   }
829
830   void solve131() {
831       init();
832       build();
833       print();
834   }
835
836   void solve132() {
837       init();
838       build();
839       print();
840   }
841
842   void solve133() {
843       init();
844       build();
845       print();
846   }
847
848   void solve134() {
849       init();
850       build();
851       print();
852   }
853
854   void solve135() {
855       init();
856       build();
857       print();
858   }
859
860   void solve136() {
861       init();
862       build();
863       print();
864   }
865
866   void solve137() {
867       init();
868       build();
869       print();
870   }
871
872   void solve138() {
873       init();
874       build();
875       print();
876   }
877
878   void solve139() {
879       init();
880       build();
881       print();
882   }
883
884   void solve140() {
885       init();
886       build();
887       print();
888   }
889
890   void solve141() {
891       init();
892       build();
893       print();
894   }
895
896   void solve142() {
897       init();
898       build();
899       print();
900   }
901
902   void solve143() {
903       init();
904       build();
905       print();
906   }
907
908   void solve144() {
909       init();
910       build();
911       print();
912   }
913
914   void solve145() {
915       init();
916       build();
917       print();
918   }
919
920   void solve146() {
921       init();
922       build();
923       print();
924   }
925
926   void solve147() {
927       init();
928       build();
929       print();
930   }
931
932   void solve148() {
933       init();
934       build();
935       print();
936   }
937
938   void solve149() {
939       init();
940       build();
941       print();
942   }
943
944   void solve150() {
945       init();
946       build();
947       print();
948   }
949
950   void solve151() {
951       init();
952       build();
953       print();
954   }
955
956   void solve152() {
957       init();
958       build();
959       print();
960   }
961
962   void solve153() {
963       init();
964       build();
965       print();
966   }
967
968   void solve154() {
969       init();
970       build();
971       print();
972   }
973
974   void solve155() {
975       init();
976       build();
977       print();
978   }
979
980   void solve156() {
981       init();
982       build();
983       print();
984   }
985
986   void solve157() {
987       init();
988       build();
989       print();
990   }
991
992   void solve158() {
993       init();
994       build();
995       print();
996   }
997
998   void solve159() {
999       init();
1000      build();
1001      print();
1002  }
1003
1004  void solve160() {
1005      init();
1006      build();
1007      print();
1008  }
1009
1010  void solve161() {
1011      init();
1012      build();
1013      print();
1014  }
1015
1016  void solve162() {
1017      init();
1018      build();
1019      print();
1020  }
1021
1022  void solve163() {
1023      init();
1024      build();
1025      print();
1026  }
1027
1028  void solve164() {
1029      init();
1030      build();
1031      print();
1032  }
1033
1034  void solve165() {
1035      init();
1036      build();
1037      print();
1038  }
1039
1040  void solve166() {
1041      init();
1042      build();
1043      print();
1044  }
1045
1046  void solve167() {
1047      init();
1048      build();
1049      print();
1050  }
1051
1052  void solve168() {
1053      init();
1054      build();
1055      print();
1056  }
1057
1058  void solve169() {
1059      init();
1060      build();
1061      print();
1062  }
1063
1064  void solve170() {
1065      init();
1066      build();
1067      print();
1068  }
1069
1070  void solve171() {
1071      init();
1072      build();
1073      print();
1074  }
1075
1076  void solve172() {
1077      init();
1078      build();
1079      print();
1080  }
1081
1082  void solve173() {
1083      init();
1084      build();
1085      print();
1086  }
1087
1088  void solve174() {
1089      init();
1090      build();
1091      print();
1092  }
1093
1094  void solve175() {
1095      init();
1096      build();
1097      print();
1098  }
1099
1100  void solve176() {
1101      init();
1102      build();
1103      print();
1104  }
1105
1106  void solve177() {
1107      init();
1108      build();
1109      print();
1110  }
1111
1112  void solve178() {
1113      init();
1114      build();
1115      print();
1116  }
1117
1118  void solve179() {
1119      init();
1120      build();
1121      print();
1122  }
1123
1124  void solve180() {
1125      init();
1126      build();
1127      print();
1128  }
1129
1130  void solve181() {
1131      init();
1132      build();
1133      print();
1134  }
1135
1136  void solve182() {
1137      init();
1138      build();
1139      print();
1140  }
1141
1142  void solve183() {
1143      init();
1144      build();
1145      print();
1146  }
1147
1148  void solve184() {
1149      init();
1150      build();
1151      print();
1152  }
1153
1154  void solve185() {
1155      init();
1156      build();
1157      print();
1158  }
1159
1160  void solve186() {
1161      init();
1162      build();
1163      print();
1164  }
1165
1166  void solve187() {
1167      init();
1168      build();
1169      print();
1170  }
1171
1172  void solve188() {
1173      init();
1174      build();
1175      print();
1176  }
1177
1178  void solve189() {
1179      init();
1180      build();
1181      print();
1182  }
1183
1184  void solve190() {
1185      init();
1186      build();
1187      print();
1188  }
1189
1190  void solve191() {
1191      init();
1192      build();
1193      print();
1194  }
1195
1196  void solve192() {
1197      init();
1198      build();
1199      print();
1200  }
1201
1202  void solve193() {
1203      init();
1204      build();
1205      print();
1206  }
1207
1208  void solve194() {
1209      init();
1210      build();
1211      print();
1212  }
1213
1214  void solve195() {
1215      init();
1216      build();
1217      print();
1218  }
1219
1220  void solve196() {
1221      init();
1222      build();
1223      print();
1224  }
1225
1226  void solve197() {
1227      init();
1228      build();
1229      print();
1230  }
1231
1232  void solve198() {
1233      init();
1234      build();
1235      print();
1236  }
1237
1238  void solve199() {
1239      init();
1240      build();
1241      print();
1242  }
1243
1244  void solve200() {
1245      init();
1246      build();
1247      print();
1248  }
1249
1250  void solve201() {
1251      init();
1252      build();
1253      print();
1254  }
1255
1256  void solve202() {
1257      init();
1258      build();
1259      print();
1260  }
1261
1262  void solve203() {
1263      init();
1264      build();
1265      print();
1266  }
1267
1268  void solve204() {
1269      init();
1270      build();
1271      print();
1272  }
1273
1274  void solve205() {
1275      init();
1276      build();
1277      print();
1278  }
1279
1280  void solve206() {
1281      init();
1282      build();
1283      print();
1284  }
1285
1286  void solve207() {
1287      init();
1288      build();
1289      print();
1290  }
1291
1292  void solve208() {
1293      init();
1294      build();
1295      print();
1296  }
1297
1298  void solve209() {
1299      init();
1300      build();
1301      print();
1302  }
1303
1304  void solve210() {
1305      init();
1306      build();
1307      print();
1308  }
1309
1310  void solve211() {
1311      init();
1312      build();
1313      print();
1314  }
1315
1316  void solve212() {
1317      init();
1318      build();
1319      print();
1320  }
1321
1322  void solve213() {
1323      init();
1324      build();
1325      print();
1326  }
1327
1328  void solve214() {
1329      init();
1330      build();
1331      print();
1332  }
1333
1334  void solve215() {
1335      init();
1336      build();
1337      print();
1338  }
1339
1340  void solve216() {
1341      init();
1342      build();
1343      print();
1344  }
1345
1346  void solve217() {
1347      init();
1348      build();
1349      print();
1350  }
1351
1352  void solve218() {
1353      init();
1354      build();
1355      print();
1356  }
1357
1358  void solve219() {
1359      init();
1360      build();
1361      print();
1362  }
1363
1364  void solve220() {
1365      init();
1366      build();
1367      print();
1368  }
1369
1370  void solve221() {
1371      init();
1372      build();
1373      print();
1374  }
1375
1376  void solve222() {
1377      init();
1378      build();
1379      print();
1380  }
1381
1382  void solve223() {
1383      init();
1384      build();
1385      print();
1386  }
1387
1388  void solve224() {
1389      init();
1390      build();
1391      print();
1392  }
1393
1394  void solve225() {
1395      init();
1396      build();
1397      print();
1398  }
1399
1400  void solve226() {
1401      init();
1402      build();
1403      print();
1404  }
1405
1406  void solve227() {
1407      init();
1408      build();
1409      print();
1410  }
1411
1412  void solve228() {
1413      init();
1414      build();
1415      print();
1416  }
1417
1418  void solve229() {
1419      init();
1420      build();
1421      print();
1422  }
1423
1424  void solve230() {
1425      init();
1426      build();
1427      print();
1428  }
1429
1430  void solve231() {
1431      init();
1432      build();
1433      print();
1434  }
1435
1436  void solve232() {
1437      init();
1438      build();
1439      print();
1440  }
1441
1442  void solve233() {
1443      init();
1444      build();
1445      print();
1446  }
1447
1448  void solve234() {
1449      init();
1450      build();
1451      print();
1452  }
1453
1454  void solve235() {
1455      init();
1456      build();
1457      print();
1458  }
1459
1460  void solve236() {
1461      init();
1462      build();
1463      print();
1464  }
1465
1466  void solve237() {
1467      init();
1468      build();
1469      print();
1470  }
1471
1472  void solve238() {
1473      init();
1474      build();
1475      print();
1476  }
1477
1478  void solve239() {
1479      init();
1480      build();
1481      print();
1482  }
1483
1484  void solve240() {
1485      init();
1486      build();
1487      print();
1488  }
1489
1490  void solve241() {
1491      init();
1492      build();
1493      print();
1494  }
1495
1496  void solve242() {
1497      init();
1498      build();
1499      print();
1500  }
1501
1502  void solve243() {
1503      init();
1504      build();
1505      print();
1506  }
1507
1508  void solve244() {
1509      init();
1510      build();
1511      print();
1512  }
1513
1514  void solve245() {
1515      init();
1516      build();
1517      print();
1518  }
1519
1520  void solve246() {
1521      init();
1522      build();
1523      print();
1524  }
1525
1526  void solve247() {
1527      init();
1528      build();
1529      print();
1530  }
1531
1532  void solve248() {
1533      init();
1534      build();
1535      print();
1536  }
1537
1538  void solve249() {
1539      init();
1540      build();
1541      print();
1542  }
1543
1544  void solve250() {
1545      init();
1546      build();
1547      print();
1548  }
1549
1550  void solve251() {
1551      init();
1552      build();
1553      print();
1554  }
1555
1556
```

```

18 }Tree[MAXN];
19 int idx; // 当前用到的节点编号
20
21 map<char, string> mp; // 编码结果
22
23 // 新建一个权值为w的节点,左右儿子节点分别为l、r
24 Node newNode(int w, int l = 0, int r = 0) {
25     return Tree[idx] = { ++idx, w, l, r };
26 }
27
28 HuffmanTree(int _n, const vector<int>& _a, const vector<char>& _chs):
29 n(_n) { // a[]下标从1开始
30     idx = 0;
31
32     priority_queue<Node, vector<Node>, greater<Node>> heap;
33     for (int i = 1; i <= n; i++) {
34         heap.push(newNode(_a[i]));
35         Tree[i].data = _chs[i];
36     }
37
38     while (heap.size() >= 2) {
39         auto left = heap.top(); heap.pop();
40         auto right = heap.top(); heap.pop();
41         auto root = newNode(left.w + right.w, left.id, right.id);
42         heap.push(root);
43         Tree[left.id].fa = Tree[right.id].fa = root.id;
44     }
45 }
46
47 void print() {
48     for (int i = 1; i <= idx; i++) {
49         cout << Tree[i].id << ' ' << Tree[i].w << ' '
50             << Tree[i].l << ' ' << Tree[i].r << endl;
51     }
52 }
53
54 void encode() {
55     for (int i = 1; i <= n; i++) { // 枚举叶子节点
56         string res;
57         for (int u = i; u != idx; u = Tree[u].fa) {
58             if (Tree[Tree[u].fa].l == u) res.push_back('0');
59             else res.push_back('1');
60         }
61
62         reverse(all(res));
63         cout << Tree[i].data << " :" << res << endl;
64         mp[Tree[i].data] = res;
65     }
66 }
67
68 string encode(string s) {
69     string res;
70     for (auto ch : s) res += mp[ch];
71     return res;
72 }
73
74 void decode(string s) {
75     int u = idx;

```

```

76     string res;
77     for (auto ch : s) {
78         if (ch == '0') {
79             if (Tree[u].l) u = Tree[u].l;
80             else {
81                 cout << "error!" << endl;
82                 return;
83             }
84         }
85         else {
86             if (Tree[u].r) u = Tree[u].r;
87             else {
88                 cout << "error!" << endl;
89                 return;
90             }
91         }
92
93         if (1 <= Tree[u].id && Tree[u].id <= n) { // 叶子节点
94             res.push_back(Tree[u].data);
95             u = idx; // 回到根节点
96         }
97     }
98     cout << (u == idx ? res : "error!") << endl;
99 }
100 };
101
102 void solve() {
103     int n; cin >> n;
104     vector<char> chs(n + 1);
105     vector<int> a(n + 1);
106     for (int i = 1; i <= n; i++) cin >> chs[i];
107     for (int i = 1; i <= n; i++) cin >> a[i];
108
109     HuffmanTree tr(n, a, chs);
110     tr.encode();
111
112     string s; cin >> s;
113     cout << tr.encode(s) << endl;
114
115     cin >> s;
116     tr.decode(s);
117 }

```

6. 最短路

6.1 Floyd算法求最短路

```

1  const int MAXN = 1e3 + 5;
2  namespace Floyd {
3      int n; // 节点数
4      ll dis[MAXN][MAXN]; // dis[u][v]表示节点u与v间的最短距离
5
6      void init() { // 初始化dis[][]
7          for (int i = 1; i <= n; i++) {
8              for (int j = 1; j <= n; j++)

```

```

9     dis[i][j] = i == j ? 0 : INF;
10 }
11 }
12
13 void floyd() {
14     for (int k = 1; k <= n; k++) {
15         for (int i = 1; i <= n; i++) {
16             for (int j = 1; j <= n; j++)
17                 dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
18         }
19     }
20 }
21 }
22 using namespace Floyd;

```

6.2 Floyd算法求传递闭包

```

1  const int MAXN = 1e3 + 5;
2  namespace Floyd {
3      int n; // 节点数
4      bool d[MAXN][MAXN]; // 图的可达矩阵:d[u][v]表示节点u与v间的是否有边
5
6      void floyd() {
7          for (int k = 1; k <= n; k++) {
8              for (int i = 1; i <= n; i++) {
9                  for (int j = 1; j <= n; j++)
10                     d[i][j] |= d[i][k] & d[k][j];
11              }
12          }
13      }
14  }
15  using namespace Floyd;
16
17  void solve() {
18      cin >> n;
19      for (int i = 1; i <= n; i++)
20          for (int j = 1; j <= n; j++) cin >> d[i][j];
21
22      floyd();
23      for (int i = 1; i <= n; i++) {
24          for (int j = 1; j <= n; j++)
25              cout << d[i][j] << " \n"[j == n];
26      }
27  }

```

6.3 Dijkstra算法

```

1  struct Dijkstra {
2      int n;
3      vector<vector<pair<int, int>>> edges;
4      vector<bool> state;
5      vector<ll> dis;
6
7      Dijkstra(int _n, const vector<vector<pair<int, int>>>& _edges) :n(_n), edges(_edges) {

```

```

8     state.resize(n + 1);
9     dis = vector<ll>(n + 1, INF);
10 }
11
12 void dijkstra(int s) {
13     dis[s] = 0;
14     for (int i = 0; i < n - 1; i++) {
15         int t = -1, j;
16         for (j = 1; j <= n; j++) {
17             if (!state[j] && (t == -1 || dis[t] > dis[j]))
18                 t = j;
19         }
20         if (j == n) break;
21
22         state[t] = true;
23         for (auto [v, w] : edges[t])
24             dis[v] = min(dis[v], dis[t] + w);
25     }
26 }
27 };

```

6.4 SPFA算法

给定一个有重边的无向图,求节点1到节点 n 的最短路.

```

1  const int MAXN = 1e3 + 5;
2  struct SPFA {
3      int n; // 节点数
4      int d[MAXN][MAXN];
5      ll dis[MAXN]; // 起点到每个节点的最短距离
6      bool state[MAXN]; // 记录当前每个节点是否在队列中
7
8      SPFA(int _n) : n(_n) {
9          memset(d, INF, sizeof(d));
10         memset(state, false, sizeof(state));
11     }
12
13     void addEdge(int u, int v, int w) {
14         d[u][v] = d[v][u] = min(d[u][v], w); // 重边保留最短的
15     }
16
17     ll spfa(int s) { // 求起点s到其他节点的最短路
18         memset(dis, INF, sizeof(dis));
19
20         queue<int> que;
21         dis[s] = 0;
22         que.push(s);
23         state[s] = true;
24
25         while (que.size()) {
26             auto u = que.front(); que.pop();
27             state[u] = false;
28
29             for (int v = 1; v <= n; v++) {
30                 if (d[u][v] != INF) {
31                     if (dis[v] > dis[u] + d[u][v]) {

```

```

32         dis[v] = dis[u] + d[u][v];
33
34         if (!state[v]) {
35             que.push(v);
36             state[v] = true;
37         }
38     }
39 }
40 }
41 }
42 return dis[n];
43 }
44 };
45
46 void solve() {
47     int n;
48     while (cin >> n) {
49         SPFA solver(n);
50
51         int m; cin >> m;
52         while (m-- > 0) {
53             int u, v, w; cin >> u >> v >> w;
54             solver.addEdge(u, v, w);
55         }
56
57         cout << solver.spfa(1) << endl;
58     }
59 }

```

7. 最小生成树

7.1 Prim算法

```

1  const int MAXN = 1e3 + 5;
2  struct Prim {
3      int n; // 节点数
4      int edges[MAXN][MAXN];
5      int dis[MAXN]; // dis[u]表示节点u到连通块的最短距离
6      bool state[MAXN]; // 记录当前每个节点是否在连通块中
7
8      Prim(int _n) : n(_n) {
9          memset(state, false, sizeof(state));
10     }
11
12     int prim() { // 求MST, 返回MST的边权和
13         memset(dis, INF, sizeof(dis));
14
15         int res = 0;
16         for (int i = 0; i < n; i++) { // 做n次迭代
17             int t = -1; // 不在连通块中的与连通块距离最近的节点
18             for (int j = 1; j <= n; j++) {
19                 if (!state[j] && (t == -1 || dis[t] > dis[j]))
20                     t = j;
21             }
22

```

```

23     if (i && dis[t] == INF) return INFF; // 图不连通,不存在MST
24
25     if (i) res += dis[t]; // 第一个节点无边,注意先更新res再更新dis[]
26     state[t] = true; // 将节点t加入连通块
27     for (int j = 1; j <= n; j++)
28         dis[j] = min(dis[j], (ll)edges[t][j]); // 注意不是dis[j]+edges[t][j]
29 }
30 return res;
31 }
32 };

```

7.2 Kruskal算法

```

1 struct kruskal {
2     int n, m; // 节点数、边数
3     struct Edge {
4         int u, v, w;
5
6         bool operator<(const Edge& p) const {
7             return w < p.w;
8         }
9     };
10    vector<Edge> edges;
11    vector<int> fa;
12
13    kruskal(int _n) : n(_n), m(0) {
14        fa.resize(n + 1);
15        iota(all(fa), 0);
16    }
17
18    int find(int x) {
19        return x == fa[x] ? x : fa[x] = find(fa[x]);
20    }
21
22    ll kruskal() { // 求MST,返回MST的边权和
23        sort(all(edges));
24
25        ll res = 0; // MST的边权和
26        int cnt = 0; // 当前连的边数
27        for (int i = 0; i < m; i++) {
28            auto [u, v, w] = edges[i];
29
30            u = find(u), v = find(v);
31            if (u != v) {
32                fa[u] = v;
33                res += w;
34                cnt++;
35            }
36        }
37        return cnt < n - 1 ? INFF : res;
38    }
39 };

```


8. 拓扑排序

给定一个DAG,用邻接矩阵存图,求拓扑序的第一个节点到其他节点的最长路,输出路径.

```

1  const double eps = 1e-8;
2
3  int sgn(double x) {
4      if (fabs(x) < eps) return 0;
5      else return x < 0 ? -1 : 1;
6  }
7
8  const int MAXN = 1e3 + 5;
9  namespace TopologicalSorting {
10     int n;
11     double edges[MAXN][MAXN];
12     int in[MAXN];
13     vector<int> res; // 拓扑序
14     double dis[MAXN];
15     int pre[MAXN];
16
17     void addEdge(int u, int v, double w) {
18         edges[u][v] = w;
19         in[v]++;
20     }
21
22     void topo() {
23         queue<int> que;
24         for (int i = 0; i < n; i++)
25             if (!in[i]) que.push(i);
26
27         while (que.size()) {
28             auto u = que.front(); que.pop();
29             res.push_back(u);
30
31             for (int v = 0; v < n; v++) {
32                 if (sgn(edges[u][v]))
33                     if (--in[v] == 0) que.push(v);
34             }
35         }
36     }
37
38     void cal() {
39         for (auto u : res) {
40             for (int v = 0; v < n; v++) {
41                 if (sgn(edges[u][v])) {
42                     double tmp = dis[u] + edges[u][v];
43                     if (dis[v] < tmp) {
44                         dis[v] = tmp;
45                         pre[v] = u;
46                     }
47                 }
48             }
49         }
50     }
51 }
52 using namespace TopologicalSorting;
53

```

```

54 void solve() {
55     int m; cin >> n >> m;
56     while (m--) {
57         int u, v; double w; cin >> u >> v >> w;
58         addEdge(u, v, w);
59     }
60
61     topo();
62     cal();
63
64     for (int i = 0; i < n; i++) {
65         if (i == 5) continue;
66
67         int cur = i;
68         vector<int> path;
69         while (true) {
70             path.push_back(cur);
71             cur = pre[cur];
72
73             if (cur == 5) {
74                 path.push_back(cur);
75                 break;
76             }
77         }
78
79         reverse(all(path));
80         for (auto u : path) cout << u << " \n"[u == path.back()];
81     }
82 }

```

9. 查找

9.1 顺序查找

实现一个顺序表,支持插入元素、删除位置、删除元素、查找元素,其中查找元素需输出比较次数.

```

1  const int MAXN = 1e5 + 5;
2  int n;
3  int a[MAXN];
4
5  void print() {
6      cout << n << ' ';
7      for (int i = 1; i <= n; i++) cout << a[i] << ' ';
8      cout << endl;
9  }
10
11 void solve() {
12     cin >> n;
13     for (int i = 1; i <= n; i++) cin >> a[i];
14     print();
15
16     int pos, val; cin >> pos >> val;
17     if (pos < 1 || pos > n + 1) cout << "ERROR" << endl;
18     else {
19         for (int i = n; i >= pos; i--) a[i + 1] = a[i];

```

```

20     n++;
21     a[pos] = val;
22     print();
23 }
24
25 cin >> pos;
26 if (pos < 1 || pos > n) cout << "ERROR" << endl;
27 else {
28     for (int i = pos; i <= n; i++) a[i] = a[i + 1];
29     n--;
30     print();
31 }
32
33 cin >> val;
34 for (pos = 1; pos <= n && a[pos] != val; pos++);
35 if (pos == n + 1) cout << "ERROR" << endl;
36 else {
37     for (int i = pos; i <= n; i++) a[i] = a[i + 1];
38     n--;
39     print();
40 }
41
42 cin >> val;
43 bool ok = false;
44 for (int i = n; i >= 1; i--) { // 注意从后往前查找
45     if (a[i] == val) {
46         cout << 1 << ' ' << i << ' ' << n - i + 1 << endl;
47         ok = true;
48         break;
49     }
50 }
51 if (!ok) cout << 0 << ' ' << 0 << ' ' << n + 1 << endl;
52 }

```

9.2 折半查找

给定一个下标从1开始的有序序列,用折半查找找到元素的位置.

```

1 void solve() {
2     int n; cin >> n;
3     vector<int> a(n + 5);
4     for (int i = 1; i <= n; i++) cin >> a[i];
5
6     CaseT{
7         int x; cin >> x;
8
9         int l = 1, r = n;
10        while (l <= r) {
11            int mid = l + r >> 1;
12            if (a[mid] > x) r = mid - 1;
13            else l = mid + 1;
14        }
15        cout << (a[r] == x ? to_string(r) : "error") << endl;
16    }
17 }

```

9.3 顺序索引查找

索引表和块内查找都采用不带哨兵、从头开始的顺序查找。

```

1 void solve() {
2     int n; cin >> n; // 序列长度
3     vector<int> a(n + 1);
4     for (int i = 1; i <= n; i++) cin >> a[i];
5     int m; cin >> m; // 分块数
6     vector<int> rangeMax(m + 1);
7     for (int i = 1; i <= m; i++) cin >> rangeMax[i];
8
9     vector<int> start(m + 2); // 每个分块的起始下标
10    start[1] = 1, start[m + 1] = n; // 边界
11    for (int i = 1, j = 1; i <= n; i++)
12        if (a[i] > rangeMax[j]) start[++j] = i;
13
14    CaseT{
15        int x; cin >> x;
16        if (x > rangeMax[m]) cout << "error" << endl;
17        else {
18            int step = 0; // 比较次数
19            int pos;
20            for (pos = 1; pos <= m; pos++) {
21                step++;
22                if (rangeMax[pos] >= x) break;
23            }
24
25            bool ok = false;
26            for (int i = start[pos]; i <= start[pos + 1]; i++) {
27                step++;
28                if (a[i] == x) {
29                    cout << i << '-' << step << endl;
30                    ok = true;
31                    break;
32                }
33            }
34            if (!ok) cout << "error" << endl;
35        }
36    }
37 }

```

9.4 哈希查找(线性探测法)

```

1 const int MAXN = 1e3 + 5;
2 struct HashTable {
3     const int MOD = 11;
4     int len; // 表长
5     int ha[MAXN];
6
7     HashTable(int _len) : len(_len) {
8         for (int i = 0; i < len; i++) ha[i] = -1;
9     }
10
11    void insert(int x) {

```

```

12     int pos = x % MOD;
13     while (pos < len && ~ha[pos]) pos = (pos + 1) % len;
14     ha[pos] = x;
15 }
16
17 pair<int, int> find(int x) { // 返回查找次数、查找成功的位置
18     int pos = x % MOD;
19     int step = 1; // 查找次数
20     for (int i = pos; i < len; i++, step++) {
21         if (ha[i] == -1) return { step, -1 };
22         else if (ha[i] == x) return { step, i + 1 };
23     }
24
25     for (int i = 0; i < pos; i++, step++) {
26         if (ha[i] == -1) return { step, -1 };
27         else if (ha[i] == x) return { step, i + 1 };
28     }
29     return { step, -1 };
30 }
31
32 void print() {
33     for (int i = 0; i < len; i++)
34         cout << (~ha[i] ? to_string(ha[i]) : "NULL") << " \n"[i == len - 1];
35 }
36 };
37
38 void solve() {
39     int len, n; cin >> len >> n;
40     HashTable hat(len);
41     while (n--) {
42         int x; cin >> x;
43         hat.insert(x);
44     }
45     hat.print();
46
47     CaseT{
48         int x; cin >> x;
49
50         auto [u, v] = hat.find(x);
51         if (~v) cout << 1 << ' ' << u << ' ' << v << endl;
52         else cout << 0 << ' ' << u << endl;
53     }
54 }

```

9.5 哈希查找(二次探测法)

```

1  const int MAXN = 1e3 + 5;
2  struct HashTable {
3      const int MOD = 11;
4      int len; // 表长
5      int ha[MAXN];
6
7      HashTable(int _len) : len(_len) {
8          for (int i = 0; i < len; i++) ha[i] = -1;
9      }
10

```

```

11 void insert(int x) {
12     int pos = x % MOD;
13     for (int cur = 0; ; cur++) {
14         for (int i = 0; i < 2; i++) {
15             int tmp = ((pos + (i ? -1 : 1) * cur * cur) % len + len) % len;
16             if (ha[tmp] == -1) {
17                 ha[tmp] = x;
18                 return;
19             }
20         }
21     }
22 }
23
24 pair<int, int> find(int x) { // 返回查找次数、查找成功的位置
25     int pos = x % MOD;
26     int step = 1; // 查找次数
27     for (int cur = 0; ; cur++) {
28         for (int i = 0; i < 2; i++) {
29             int tmp = ((pos + (i ? -1 : 1) * cur * cur) % len + len) % len;
30             if (cur) step++; // cur=0时只需一次
31
32             if (ha[tmp] == -1) return { step, -1 };
33             else if (ha[tmp] == x) return { step, tmp + 1 };
34         }
35
36         if (cur > sqrt(len)) return { step, -1 };
37     }
38 }
39
40 void print() {
41     for (int i = 0; i < len; i++)
42         cout << (ha[i] ? to_string(ha[i]) : "NULL") << " \n"[i == len - 1];
43 }
44 };
45
46 void solve() {
47     int len, n; cin >> len >> n;
48     HashTable hat(len);
49     while (n--) {
50         int x; cin >> x;
51         hat.insert(x);
52     }
53     hat.print();
54
55     CaseT{
56         int x; cin >> x;
57
58         auto [u, v] = hat.find(x);
59         if (~v) cout << 1 << ' ' << u << ' ' << v << endl;
60         else cout << 0 << ' ' << u << endl;
61     }
62 }

```

9.6 哈希查找(链地址法)

```

1  const int MAXN = 1e3 + 5;
2  struct HashTable {
3      const int MOD = 11;
4      list<int> ha[MAXN];
5
6      void insert(int x) {
7          int pos = x % MOD;
8          ha[pos].insert(ha[pos].begin(), x);
9      }
10
11     pair<int, int> find(int x) { // 返回查找次数、查找成功的位置
12         int pos = x % MOD;
13         int step = 1;
14         for (auto it = ha[pos].begin(); it != ha[pos].end(); it++, step++)
15             if (*it == x) return { step, pos };
16         return { -1, -1 };
17     }
18 };
19
20 void solve() {
21     HashTable hat;
22     int n; cin >> n;
23     while (n--) {
24         int x; cin >> x;
25         hat.insert(x);
26     }
27
28     CaseT{
29         int x; cin >> x;
30
31         auto [u, v] = hat.find(x);
32         if (~v) cout << v << ' ' << u << endl;
33         else {
34             cout << "error" << endl;
35             hat.insert(x);
36         }
37     }
38 }

```

10. 排序

10.1 直插排序

升序排列.

```

1  const int MAXN = 1e3 + 5;
2  int n;
3  int a[MAXN];
4
5  void print() {
6      for (int i = 0; i < n; i++)
7          cout << a[i] << " \n"[i == n - 1];

```

```

8 }
9
10 void insertSort() {
11     for (int i = 1; i < n; i++) {
12         for (int j = i - 1; j >= 0 && a[j] > a[j + 1]; j--)
13             swap(a[j], a[j + 1]);
14         print();
15     }
16 }
17
18 void solve() {
19     cin >> n;
20     for (int i = 0; i < n; i++) cin >> a[i];
21
22     insertSort();
23 }

```

10.2 希尔排序

降序排列.

```

1  const int MAXN = 1e3 + 5;
2  int n;
3  int a[MAXN];
4
5  void print() {
6      for (int i = 0; i < n; i++)
7          cout << a[i] << " \n"[i == n - 1];
8  }
9
10 void shellSort() {
11     int gap = n;
12     while (gap > 1) {
13         gap /= 2;
14         for (int i = gap; i < n; i++) {
15             int tmp;
16             if (a[i - gap] < a[i]) {
17                 tmp = a[i];
18                 int j = i - gap;
19                 while (j >= 0 && a[j] < tmp) {
20                     a[j + gap] = a[j];
21                     j -= gap;
22                 }
23                 a[j + gap] = tmp;
24             }
25         }
26         print();
27     }
28 }
29
30 void solve() {
31     cin >> n;
32     for (int i = 0; i < n; i++) cin >> a[i];
33
34     shellSort();
35     cout << endl;

```


10.3 冒泡排序

升序排列.

```

1  const int MAXN = 1e3 + 5;
2  int n;
3  int a[MAXN];
4  int ans; // 交换次数
5
6  void print() {
7      for (int i = 0; i < n; i++)
8          cout << a[i] << " \n"[i == n - 1];
9  }
10
11 void bubbleSort() {
12     for (int i = 0; i < n - 1; i++) {
13         for (int j = 0; j < n - i - 1; j++) {
14             if (a[j] > a[j + 1]) {
15                 swap(a[j], a[j + 1]);
16                 ans++;
17             }
18         }
19     }
20 }
21
22 void solve() {
23     while (cin >> n) {
24         for (int i = 0; i < n; i++) cin >> a[i];
25
26         ans = 0;
27         bubbleSort();
28         cout << ans << endl;
29     }
30 }

```

10.4 快速排序

升序排列.

```

1  const int MAXN = 1e3 + 5;
2  int n;
3  int a[MAXN];
4
5  void print() {
6      for (int i = 1; i <= n; i++)
7          cout << a[i] << " \n"[i == n];
8  }
9
10 void quickSort(int l, int r) {
11     if (l >= r) return;
12
13     int pivot = a[l], st = l, ed = r;

```

```

14 while (l < r) {
15     for (int i = r; i > l; i--, r--) {
16         if (a[i] < pivot) {
17             a[l++] = a[i];
18             break;
19         }
20     }
21
22     for (int i = l; i < r; i++, l++) {
23         if (a[i] > pivot) {
24             a[r--] = a[i];
25             break;
26         }
27     }
28 }
29
30 a[l] = pivot;
31 print();
32
33 quickSort(st, l - 1), quickSort(l + 1, ed);
34 }
35
36 void solve() {
37     cin >> n;
38     for (int i = 1; i <= n; i++) cin >> a[i];
39
40     quickSort(1, n);
41     cout << endl;
42 }

```

输出每趟排好序的元素及其位置.

```

1  const int MAXN = 1e3 + 5;
2  int n;
3  int a[MAXN];
4
5  int partition(int l, int r) {
6      int pivot = a[l];
7      while (l < r) {
8          while (l < r && a[r] >= pivot) r--;
9          a[l] = a[r];
10         while (l < r && a[l] < pivot) l++;
11         a[r] = a[l];
12     }
13     a[l] = pivot;
14
15     cout << pivot << ' ' << l << endl;
16     return l;
17 }
18
19 void quickSort(int l, int r) {
20     if (l <= r) {
21         int pos = partition(l, r);
22         quickSort(l, pos - 1), quickSort(pos + 1, r);
23     }

```

```

24 }
25
26 void solve() {
27     cin >> n;
28     for (int i = 1; i <= n; i++) cin >> a[i];
29
30     quickSort(1, n);
31     cout << endl;
32 }

```

10.5 堆排序

降序排列,建小根堆.

```

1  const int MAXN = 1e3 + 5;
2  struct HeapSort {
3      int n;
4      int heap[MAXN];
5
6      HeapSort(int _n) : n(_n) {
7          for (int i = 0; i < n; i++) cin >> heap[i];
8
9          init();
10         heapSort();
11     }
12
13     void sift(int u, int length) {
14         int ls = u * 2 + 1, rs = u * 2 + 2;
15         if (ls >= length) return; // 不存在左儿子
16
17         if (rs < length) { // 存在右儿子
18             if (heap[ls] < heap[rs]) {
19                 if (heap[ls] < heap[u]) {
20                     swap(heap[ls], heap[u]);
21                     sift(ls, length);
22                 }
23             }
24             else {
25                 if (heap[rs] < heap[u]) {
26                     swap(heap[rs], heap[u]);
27                     sift(rs, length);
28                 }
29             }
30         }
31         else { // 只有左儿子
32             if (heap[ls] < heap[u]) {
33                 swap(heap[ls], heap[u]);
34                 sift(ls, length);
35             }
36         }
37     }
38
39     void print() {
40         cout << n << ' ';
41         for (int i = 0; i < n; i++)
42             cout << heap[i] << " \n"[i == n - 1];

```

```

43     }
44
45     void init() { // 调整为初始堆
46         for (int i = n / 2; i >= 0; i--) sift(i, n);
47         print();
48     }
49
50     void heapSort() {
51         for (int i = n - 1; i >= 1; i--) {
52             swap(heap[0], heap[i]);
53             sift(0, i);
54             print();
55         }
56     }
57 };

```

10.6 选择排序

升序排列.

```

1  const int MAXN = 1e3 + 5;
2  int n;
3  int a[MAXN];
4
5  void print() {
6      for (int i = 1; i <= n; i++)
7          cout << a[i] << " \n"[i == n];
8  }
9
10 void selectSort() {
11     for (int i = 1; i <= n; i++) {
12         int minidx = i;
13         for (int j = i + 1; j <= n; j++)
14             if (a[j] < a[minidx]) minidx = j;
15         swap(a[i], a[minidx]);
16         print();
17     }
18 }
19
20 void solve() {
21     cin >> n;
22     for (int i = 1; i <= n; i++) cin >> a[i];
23
24     selectSort();
25     cout << endl;
26 }

```

10.7 归并排序

```

1  const int MAXN = 1e3 + 5;
2  int n;
3  string a[MAXN], tmp[MAXN];
4
5  void print() {

```

```

6   for (int i = 0; i < n; i++)
7       cout << a[i] << " \n"[i == n - 1];
8   }
9
10  void mergeSort() {
11      int lim; // >=n的最小的2的幂次
12      for (lim = 1; lim < n; lim *= 2);
13
14      for (int len = 2; len <= lim; len *= 2) { // len为当前归并的长度
15          for (int i = 0; i < n; i += len) {
16              int l = i, r = i + len / 2;
17              if (r > n) break;
18
19              int mid = r - 1;
20              int idx = l;
21              while (l <= mid && r < min(i + len, n)) {
22                  if (a[l] > a[r]) tmp[idx++] = a[l++];
23                  else tmp[idx++] = a[r++];
24              }
25
26              while (l <= mid) tmp[idx++] = a[l++];
27              while (r < min(i + len, n)) tmp[idx++] = a[r++];
28
29              for (int j = i; j < min(i + len, n); j++)
30                  a[j] = tmp[j];
31          }
32          print();
33      }
34  }
35
36  void solve() {
37      cin >> n;
38      for (int i = 0; i < n; i++) cin >> a[i];
39
40      mergeSort();
41      cout << endl;
42  }

```

10.8 基数排序

```

1   const int MAXN = 1e3 + 5;
2   int n;
3   int a[MAXN];
4
5   void print(vector<int> a[10]) {
6       for (int i = 0; i < 10; i++) {
7           cout << i << ':';
8           if (!a[i].size()) {
9               cout << "NULL" << endl;
10              continue;
11          }
12
13          for (int j = 0; j < a[i].size(); j++) {
14              cout << "->";
15              cout << a[i][j];
16              if (j == a[i].size() - 1) cout << "->^" << endl;

```

```

17     }
18 }
19 }
20
21 void print() {
22     for (int i = 1; i <= n; i++) cout << a[i] << " \n"[i == n];
23 }
24
25 void radixSort() {
26     int lim = 0;
27     for (int i = 1; i <= n; i++)
28         while ((int)pow(10, lim) <= a[i]) lim++;
29
30     for (int i = 0; i < lim; i++) {
31         vector<int> radix[10];
32         for (int j = 1; j <= n; j++)
33             radix[a[j] / (int)pow(10, i) % 10].push_back(a[j]);
34         print(radix);
35
36         int idx = 1;
37         for (int j = 0; j < 10; j++)
38             for (auto ai : radix[j]) a[idx++] = ai;
39         print();
40     }
41 }
42
43 void solve() {
44     cin >> n;
45     for (int i = 1; i <= n; i++) cin >> a[i];
46
47     radixSort();
48     cout << endl;
49 }

```

11. 顺序表

```

1  template<class T>
2  class Array {
3  private:
4      int capacity; // 最大长度
5      int length; // 当前长度
6      T* arr;
7
8  public:
9      Array() : capacity(100), length(0), arr(nullptr) {}
10
11     ~Array() {
12         if (arr) delete[] arr;
13     }
14
15     void setCapacity(int n) { // 设置最大容量
16         T* tmp = new T[capacity = n];
17         memcpy(tmp, arr, sizeof(T) * length);
18         delete[] arr;
19         arr = tmp;

```

```

20     }
21
22     int getLength() { // 获取当前的元素个数
23         return length;
24     }
25
26     T& operator[](int idx) { // 支持通过[]访问顺序表元素,默认下标合法
27         return arr[idx];
28     }
29
30     void create(int n, int maxLength = 0) { // 创建一个长度为n的顺序表,最大长度为maxLength
31         capacity = maxLength ? maxLength : 2 * n;
32         arr = new T[capacity];
33         length = n;
34     }
35
36     void input() {
37         for (int i = 0; i < length; i++) cin >> arr[i];
38     }
39
40     void print() {
41         for (int i = 0; i < length; i++) cout << arr[i] << ' ';
42     }
43
44     bool insertToPos(T val, int pos) { // 在下标pos位置插入元素val
45         if (pos < 0 || pos > length || length == capacity) return false;
46
47         for (int i = length - 1; i >= pos; i--) arr[i + 1] = arr[i];
48         arr[pos] = val;
49         length++;
50         return true;
51     }
52
53     bool removeByPos(int pos) { // 删除下标pos的元素
54         if (pos < 0 || pos >= length) return false;
55
56         for (int i = pos; i < length; i++) arr[i] = arr[i + 1];
57         length--;
58         return true;
59     }
60 };
61
62 void print(Array<int>& arr) {
63     cout << arr.getLength() << ' ';
64     arr.print();
65     cout << endl;
66 }

```

12. 单链表

```

1 #define npt nullptr
2
3 class Node {
4 private:
5     int data;

```

```

6     Node* nxt;
7
8 public:
9     friend class LinkedList;
10
11     Node(int _data = 0) :data(_data) {
12         nxt = npt;
13     }
14
15     int getData() {
16         return data;
17     }
18 };
19
20 class LinkedList {
21 private:
22     Node* head; // 头节点,为链表的第0个节点
23     int length; // 当前链表长度(即元素个数,不含头节点)
24
25 public:
26     LinkedList(int _length = 0) :length(_length) {
27         head = new Node;
28         if (length) create(length);
29     }
30
31     ~LinkedList() {
32         Node* cur = head->nxt;
33         while (cur) {
34             Node* trash = cur;
35             cur = cur->nxt;
36             delete trash;
37         }
38     }
39
40     int getLength() { // 获取当前的元素个数
41         return length;
42     }
43
44     void create(int n) { // 新建一个长度为n的单链表,尾插法
45         length = n;
46         Node* cur = head;
47         while (n--) {
48             Node* tmp = new Node;
49             cin >> tmp->data;
50             cur->nxt = tmp;
51             cur = tmp;
52         }
53     }
54
55     void create(const vector<int>& a) { // a[]的下标从0开始
56         length = a.size();
57         Node* cur = head;
58         for (auto ai : a) {
59             Node* tmp = new Node(ai);
60             cur->nxt = tmp;
61             cur = tmp;
62         }
63     }

```



```

64
65 void print() { // 打印链表
66     vector<int> res;
67     for (Node* cur = head->nxt; cur; cur = cur->nxt)
68         res.push_back(cur->data);
69
70     for (int i = 0; i < length; i++)
71         cout << res[i] << ' ';
72     cout << endl;
73 }
74
75 Node* operator[](int pos) { // 查找第pos个节点
76     if (pos < 0 || pos > length) return npt;
77
78     Node* cur = head;
79     for (int cnt = 0; cnt < pos; cnt++, cur = cur->nxt);
80     return cur;
81 }
82
83 // 在第pos个节点之后插入一个数据域为data的节点,返回是否插入成功
84 bool insertToPos(int data, int pos) { // pos=0表示插入到链表头(头节点之后)
85     Node* cur = (*this)[pos]; // 第pos个节点
86     if (!cur) return false;
87
88     Node* tmp = new Node(data);
89     tmp->nxt = cur->nxt;
90     cur->nxt = tmp;
91     length++;
92     return true;
93 }
94
95 bool removeByPos(int pos) { // 删除第pos个节点
96     Node* cur = (*this)[pos - 1]; // 欲删除的节点的前驱节点
97     if (!cur || !(cur->nxt)) return false;
98
99     Node* trash = cur->nxt;
100     cur->nxt = trash->nxt;
101     delete trash;
102     length--;
103     return true;
104 }
105 };

```