

Competitive Programming Solution Manual

Pascal Garcia

February 4, 2016

1 UVA 195: Anagram

```
1 #include <stdio>
2 #include <cstring>
3 #include <algorithm>
4 #include <cctype>
5
6 using namespace std;
7
8 bool comparator(char a, char b)
9 {
10     if (tolower(a) == tolower(b))
11     {
12         return a<b;
13     }
14     return tolower(a) < tolower(b);
15 }
16
17 int main()
18 {
19     char word[1000];
20     int n;
21
22     scanf("%d", &n);
23     while (n--)
24     {
25         scanf("_%s", word);
26
27         sort(word, word+strlen(word), comparator);
28         printf("%s\n", word);
29         while (next_permutation(word, word+strlen(word), comparator))
30         {
31             printf("%s\n", word);
32         }
33     }
34     return 0;
35 }
```

2 UVA 608: Counterfeit Dollar

```
1 #include <stdio>
2 #include <cstring>
3 #include <climits>
4 #include <stdlib>
5
6 char plateau1[13];
7 char plateau2[13];
8 char scale[13];
9 int coins[12];
10
11 #define DOLLAR INT_MAX
12
13 void update_up(char plateau[13], int i)
14 {
```

```

15     int index = plateau[i] - 'A';
16     if (coins[index] != DOLLAR) coins[index]--;
17 }
18
19 void update_down(char plateau[13], int i)
20 {
21     int index = plateau[i] - 'A';
22     if (coins[index] != DOLLAR) coins[index]++;
23 }
24
25 void deduction()
26 {
27     int i = 0;
28     while (plateau1[i])
29     {
30         if (!strcmp(scale, "even"))
31         {
32             coins[plateau1[i] - 'A'] = DOLLAR;
33             coins[plateau2[i] - 'A'] = DOLLAR;
34         }
35         else if (!strcmp(scale, "up"))
36         {
37             update_up(plateau2, i);
38             update_down(plateau1, i);
39         }
40         else if (!strcmp(scale, "down"))
41         {
42             update_down(plateau2, i);
43             update_up(plateau1, i);
44         }
45         i++;
46     }
47 }
48
49 int main() {
50     int T;
51     scanf("%d", &T);
52     while (T--)
53     {
54         memset(coins, 0, sizeof coins);
55         for (int i=0; i < 3; ++i)
56         {
57             scanf("%s%s%s", plateau1, plateau2, scale);
58             deduction();
59         }
60         int index = -1, max = 0;
61         for (int i = 0; i < 12; i++)
62         {
63             if (coins[i] != DOLLAR && abs(coins[i]) >= max)
64             {
65                 index = i;
66                 max = abs(coins[i]);
67             }
68         }
69         printf("%c is the counterfeit coin and it is %s\n", 'A' + index, (coins[index] < 0
? "light." : "heavy."));
70     }

```

```
71     return 0;
72 }
```

3 UVA 11057: Exact Sum

```
1  #include <cstdio>
2  #include <algorithm>
3  #include <vector>
4  #include <climits>
5
6  using namespace std;
7
8  int main(int argc, char *argv[])
9  {
10     vector<int> prices(10001);
11     int nb_books;
12     while (scanf("%d", &nb_books) != EOF)
13     {
14         prices.clear();
15         while (nb_books--)
16         {
17             int v;
18             scanf("%d", &v);
19             prices.push_back(v);
20         }
21         int money;
22         scanf("%d", &money);
23         sort(prices.begin(), prices.end());
24         int price1, price2;
25         int best = INT_MAX;
26         for (vector<int>::iterator it = prices.begin(); it != prices.end(); it++)
27         {
28             int target = money - *it;
29             if (target < 0) break;
30             if (binary_search(it + 1, prices.end(), target))
31             {
32                 int delta = abs(target - *it);
33                 if (delta < best)
34                 {
35                     best = delta;
36                     price1 = min(*it, target);
37                     price2 = max(*it, target);
38                 }
39             }
40         }
41         printf("Peter should buy books whose prices are %d and %d.\n\n", price1, price2);
42     }
43     return 0;
44 }
```

4 UVA 10038: Jolly Jumpers

```
1  #include <cstdio>
```

```

2  #include <cstdlib>
3  #include <vector>
4
5  using namespace std;
6
7  int main(int argc, char *argv[])
8  {
9      int n;
10     while (scanf("%d", &n) != EOF)
11     {
12         vector<bool> sequence(n, false);
13         int prev, current;
14         scanf("%d", &prev);
15         int remaining = n-1;
16         int cpt = 0;
17         while (remaining-->0)
18         {
19             scanf("%d", &current);
20             int v = abs(prev - current);
21             if (v < n && v != 0 && !sequence[v])
22             {
23                 sequence[v] = true;
24                 cpt++;
25             }
26             prev = current;
27         }
28         if (cpt == n - 1)
29         {
30             printf("Jolly\n");
31         }
32         else
33         {
34             printf("Not jolly\n");
35         }
36     }
37     return 0;
38 }

```

5 UVA 11040: Add Bricks in the Wall

```

1  #include <stdio>
2  #include <cstring>
3
4  using namespace std;
5
6  int bricks[9][9];
7
8  void fill()
9  {
10     for (int i = 0; i < 7; i += 2)
11     {
12         bricks[0][i+1] = (bricks[2][i] - bricks[0][i] - bricks[0][i+2]) / 2;
13     }
14     for (int i = 0; i < 8; ++i)
15     {

```

```

16     for (int j = 0; j < 8; ++j)
17     {
18         bricks[i+1][j] = bricks[i][j] + bricks[i][j+1];
19     }
20 }
21 }
22
23 void display()
24 {
25     for (int i = 8; i >= 0; --i)
26     {
27         printf("%d", bricks[i][0]);
28         for (int j = 1; j < 9 - i; ++j)
29         {
30             printf("□%d", bricks[i][j]);
31         }
32         printf("\n");
33     }
34 }
35
36 int main(int argc, char *argv[])
37 {
38     int TC;
39     scanf("%d", &TC);
40     while (TC--)
41     {
42         int cpt = 0;
43         memset(bricks, 0, sizeof bricks);
44         while (cpt < 6)
45         {
46             scanf("%*d");
47             cpt++;
48         }
49         int i = 0;
50         while (cpt < 10)
51         {
52             scanf("%d", &bricks[2][i]);
53             i+= 2;
54             cpt++;
55         }
56         i = 0;
57         while (cpt++ < 15)
58         {
59             scanf("%d", &bricks[0][i]);
60             i+= 2;
61         }
62         fill();
63         display();
64     }
65     return 0;
66 }

```

6 UVA 11173: Grey Codes

```

1 #include <stdio>

```

```

2
3 using namespace std;
4
5 int main(int argc, char *argv[])
6 {
7     int TC;
8     scanf("%d", &TC);
9     while (TC--)
10    {
11        unsigned int n, k;
12        scanf("%d_%d", &n, &k);
13        printf("%d\n", k ^ (k >> 1));
14    }
15    return 0;
16 }

```

7 UVA 1203: Argus

```

1 include <cstdio>
2 include <vector>
3 include <queue>
4
5 using namespace std;
6
7 class comparison
8 {
9     public:
10    bool operator() (const pair<int, int>& p1, const pair<int, int>& p2) const
11    {
12        return p1.first > p2.first || (p1.first == p2.first && p1.second > p2.second);
13    }
14 };
15
16 int main(int argc, char *argv[])
17 {
18     char s[1024];
19     vector<int> id_to_period(3001);
20     priority_queue<pair<int, int>, vector<pair<int, int>>, comparison> pq;
21     while (scanf("%s", s), s[0] != '#')
22     {
23         int id, period;
24         scanf("%d_%d", &id, &period);
25         id_to_period[id] = period;
26         pq.push(make_pair(period, id));
27     }
28     int K;
29     scanf("%d", &K);
30     while (K--)
31     {
32         pair<int, int> item = pq.top();
33         pq.pop();
34         printf("%d\n", item.second);
35         pq.push(make_pair(item.first + id_to_period[item.second], item.second));
36     }
37     return 0;

```

8 UVA 11134: Fable Rooks

```

1 struct range
2 {
3     int left, right, id;
4     bool operator<(const range& range) const
5     {
6         return right > range.right;
7     }
8 };
9
10 const int MAX_N = 5010;
11
12 bool solve(int N, vector<range> ranges[MAX_N], int coord[MAX_N])
13 {
14     priority_queue<range> pq;
15     for (int i = 1; i <= N; ++i)
16     {
17         for_each(ranges[i].begin(), ranges[i].end(), [&pq](const range& r){ pq.push(r);
18     });
19     if (pq.size() == 0) return false;
20     range r = pq.top();
21     pq.pop();
22     if (r.right < i) return false;
23     coord[r.id] = i;
24     }
25     return true;
26 }
27
28 int main(int argc, char *argv[])
29 {
30     int N;
31     vector<range> xrange[MAX_N];
32     vector<range> yrange[MAX_N];
33     int rook_x[MAX_N];
34     int rook_y[MAX_N];
35     while (scanf("%d", &N), N)
36     {
37         range x;
38         range y;
39         for (int i = 1; i <= N; ++i)
40         {
41             xrange[i].clear();
42             yrange[i].clear();
43         }
44         for (int i = 0; i < N; ++i)
45         {
46             scanf("%d%d%d%d", &x.left, &y.left, &x.right, &y.right);
47             x.id = i;
48             y.id = i;
49             xrange[x.left].push_back(x);
50             yrange[y.left].push_back(y);

```



```

51     bool possible = solve(N, xrange, rook_x) && solve(N, yrange, rook_y);
52     if (possible)
53     {
54         for (int i = 0; i < N; ++i)
55         {
56             printf("%d_%d\n", rook_x[i], rook_y[i]);
57         }
58     }
59     else printf("IMPOSSIBLE\n");
60 }
61 return 0;
62 }

```

9 UVA 10507: Waking Up Brain

```

1  #include <cstdio>
2  #include <vector>
3  #include <string>
4  #include <sstream>
5  #include <queue>
6
7  class graph
8  {
9      public:
10     typedef std::vector<std::pair<int, float>> NEIGHBORS;
11     typedef std::vector<NEIGHBORS> ADJ_LIST;
12
13     private:
14     const int nb_vertices;
15     int nb_edges;
16     ADJ_LIST adj;
17     bool directed;
18
19     public:
20     graph(int n = 1000, bool directed = false) : nb_vertices(n), nb_edges(0), directed(
directed)
21     {
22         adj.assign(n, NEIGHBORS());
23     }
24
25     int vertices()
26     {
27         return nb_vertices;
28     }
29
30     int edges()
31     {
32         return nb_edges;
33     }
34
35     int degree(int i)
36     {
37         return adj[i].size();
38     }
39

```

```

40 void add_edge(int i, int j, float weight = 1)
41 {
42     nb_edges++;
43     adj[i].push_back(std::make_pair(j, weight));
44     if (!directed) adj[j].push_back(std::make_pair(i, weight));
45 }
46
47 const NEIGHBORS& neighbors(int i)
48 {
49     return adj[i];
50 }
51
52 bool connected(int i, int j)
53 {
54     for (const auto& edge : adj[i])
55     {
56         if (edge.first == j) return true;
57     }
58     return false;
59 }
60
61 std::string to_string()
62 {
63     std::stringstream res;
64     res << nb_vertices << " vertices, " << nb_edges << " edges\n";
65     for (int i = 0; i < nb_vertices; i++)
66     {
67         res << i << ": ";
68         for (const auto& edge : adj[i])
69         {
70             res << "(" << edge.first << ", w: " << edge.second << ") ";
71         }
72         res << "\n";
73     }
74     return res.str();
75 }
76 };
77
78 using namespace std;
79
80 #define CHAR_TO_ZONE(C) ((C) - 'A')
81
82 int main()
83 {
84     for (int slept_areas, connections; scanf("%d%d", &slept_areas, &connections) == 2;)
85     {
86         char z1, z2, z3;
87         int activated = 0;
88         vector<int> zone_nb_activation(26, 0);
89         queue<pair<int, int>> zone_activation_time;
90
91         scanf("%c%c%c", &z1, &z2, &z3);
92
93         zone_activation_time.push(make_pair(CHAR_TO_ZONE(z1), 0));
94         zone_nb_activation[CHAR_TO_ZONE(z1)] = 3;
95         zone_activation_time.push(make_pair(CHAR_TO_ZONE(z2), 0));
96         zone_nb_activation[CHAR_TO_ZONE(z2)] = 3;

```

```

97     zone_activation_time.push(make_pair(CHAR_TO_ZONE(z3), 0));
98     zone_nb_activation[CHAR_TO_ZONE(z3)] = 3;
99
100    graph g(26);
101
102    while (connections--)
103    {
104        scanf("%c%c", &z1, &z2);
105        g.add_edge(CHAR_TO_ZONE(z1), CHAR_TO_ZONE(z2));
106    }
107
108    int time = 0;
109    while (!zone_activation_time.empty())
110    {
111        auto& zone = zone_activation_time.front();
112        time = zone.second;
113        zone_activation_time.pop();
114        activated++;
115        for (auto& p : g.neighbors(zone.first))
116        {
117            if (++zone_nb_activation[p.first] == 3) zone_activation_time.push(make_pair(p.
first, zone.second + 1));
118        }
119    }
120    if (activated == slept_areas) printf("WAKE_UP_IN_%d_YEARS\n", time);
121    else printf("THIS_BRAIN_NEVER_WAKES_UP\n");
122 }
123 return 0;
124 }

```

10 UVA 793: Network Connections

```

1  #include <cstdio>
2  #include <vector>
3
4  class union_find
5  {
6  private:
7      std::vector<int> id; // id[i] = parent of i
8      std::vector<int> sz; // sz[i] = number of objects in subtree rooted at i
9      int count; // number of components
10 public:
11     union_find(int N)
12     {
13         count = N;
14         id.assign(N, 1);
15         sz.assign(N, 0);
16         for (int i = 0; i < N; i++)
17         {
18             id[i] = i;
19             sz[i] = 1;
20         }
21     }
22
23     int nb_components()

```

```

24     {
25         return count;
26     }
27
28     int size_set(int i)
29     {
30         return sz[find_set(i)];
31     }
32
33     int find_set(int i)
34     {
35         return (id[i] == i) ? i : (id[i] = find_set(id[i]));
36     }
37
38     bool connected(int i, int j)
39     {
40         return find_set(i) == find_set(j);
41     }
42
43     void union_set(int p, int q)
44     {
45         int i = find_set(p);
46         int j = find_set(q);
47         if (i == j) return;
48         // make smaller root point to larger one
49         if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
50         else { id[j] = i; sz[i] += sz[j]; }
51         count--;
52     }
53 };
54
55 using namespace std;
56
57 int main()
58 {
59     int TC;
60     scanf("%d", &TC);
61     while (TC--)
62     {
63         int N;
64         scanf("%d%c", &N);
65         union_find uf(N+1);
66         int total = 0;
67         int successful = 0;
68         char c = getchar();
69         while (c != EOF && c != '\n')
70         {
71             int a, b;
72             scanf("%d_%d%c", &a, &b);
73             if (c == 'c')
74             {
75                 uf.union_set(a, b);
76             }
77             else
78             {
79                 total++;
80                 if (uf.connected(a, b))

```

```

81     {
82         successful++;
83     }
84 }
85 c = getchar();
86 }
87 printf("%d,%d\n", successful, total - successful);
88 if (TC) printf("\n");
89 }
90 return 0;
91 }

```

11 UVA 10583: Ubiquitous Religions

```

1  class union_find
2  {
3  private:
4      std::vector<int> id; // id[i] = parent of i
5      std::vector<int> sz; // sz[i] = number of objects in subtree rooted at i
6      int count; // number of components
7  public:
8      union_find(int N)
9      {
10         count = N;
11         id.assign(N, 1);
12         sz.assign(N, 0);
13         for (int i = 0; i < N; i++)
14         {
15             id[i] = i;
16             sz[i] = 1;
17         }
18     }
19
20     int nb_components()
21     {
22         return count;
23     }
24
25     int size_set(int i)
26     {
27         return sz[find_set(i)];
28     }
29
30     int find_set(int i)
31     {
32         return (id[i] == i) ? i : (id[i] = find_set(id[i]));
33     }
34
35     bool connected(int i, int j)
36     {
37         return find_set(i) == find_set(j);
38     }
39
40     void union_set(int p, int q)
41     {

```

```

42     int i = find_set(p);
43     int j = find_set(q);
44     if (i == j) return;
45     // make smaller root point to larger one
46     if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
47     else { id[j] = i; sz[i] += sz[j]; }
48     count--;
49 }
50 };
51
52 using namespace std;
53
54 int main(int argc, char *argv[])
55 {
56     int N, M, cpt = 1;
57     while (scanf("%d%d", &N, &M), (N || M))
58     {
59         union_find uf(N);
60         while (M--)
61         {
62             int a, b;
63             scanf("%d%d", &a, &b);
64             uf.union_set(a-1, b-1);
65         }
66         printf("Case %d: %d\n", cpt++, uf.nb_components());
67     }
68     return 0;
69 }

```

12 UVA 11987: Almost Union Find

```

1  #include <cstdio>
2  #include <vector>
3
4  class union_find
5  {
6  private:
7      std::vector<int> id; // id[i] = parent of i
8      std::vector<int> sz; // sz[i] = number of objects in subtree rooted at i
9      std::vector<long> sum; // sum[i] = sum of elements in subtree rooted at i
10     int count; // number of components
11     int N;
12 public:
13     union_find(int N)
14     {
15         this->N = N;
16         count = N;
17         id.assign(2 * N + 1, 0);
18         sz.assign(N + 1, 1);
19         sum.assign(N + 1, 0);
20         for (int i = 1; i <= N; i++)
21         {
22             id[i] = id[N + i] = N + i;
23             sum[i] = i;
24         }

```

```

25     }
26
27     int nb_components()
28     {
29         return count;
30     }
31
32     int size_set(int i)
33     {
34         return sz[find_set(i) - N];
35     }
36
37     int sum_set(int i)
38     {
39         return sum[find_set(i) - N];
40     }
41
42     int find_set(int i)
43     {
44         return (id[i] == i) ? i : (id[i] = find_set(id[i]));
45     }
46
47     bool connected(int i, int j)
48     {
49         return find_set(i) == find_set(j);
50     }
51
52     void union_set(int p, int q)
53     {
54         int i = find_set(p);
55         int j = find_set(q);
56         if (i == j) return;
57         // make smaller root point to larger one
58         if (sz[i - N] < sz[j - N]) { id[i] = j; sz[j - N] += sz[i - N]; sum[j - N] += sum[i
- N]; }
59         else { id[j] = i; sz[i - N] += sz[j - N]; sum[i - N] += sum[j
- N]; }
60         count--;
61     }
62
63     void move(int p, int q)
64     {
65         int i = find_set(p);
66         int j = find_set(q);
67         if (i == j) return;
68         id[p] = j;
69         sum[j - N] += p; sz[j - N]++;
70         sum[i - N] -= p; sz[i - N]--;
71     }
72 };
73
74 using namespace std;
75
76 int main(int argc, char *argv[])
77 {
78     int N, M;
79     while (scanf("%d_%d", &N, &M) != EOF)

```

```

80     {
81         union_find uf(N);
82         while (M--)
83         {
84             int command, p, q;
85             scanf("%d", &command);
86             if (command == 1) { scanf("%d_%d", &p, &q); uf.union_set(p, q); }
87             else if (command == 2) { scanf("%d_%d", &p, &q); uf.move(p, q); }
88             else { scanf("%d", &p); printf("%d_%d\n", uf.size_set(p), uf.sum_set(p)); }
89         }
90     }
91     return 0;
92 }

```

13 UVA 11690: Money Matters

```

1  class union_find
2  {
3  private:
4      std::vector<int> id; // id[i] = parent of i
5      std::vector<int> sz; // sz[i] = number of objects in subtree rooted at i
6      int count; // number of components
7  public:
8      union_find(int N)
9      {
10         count = N;
11         id.assign(N, 1);
12         sz.assign(N, 0);
13         for (int i = 0; i < N; i++)
14         {
15             id[i] = i;
16             sz[i] = 1;
17         }
18     }
19
20     int nb_components()
21     {
22         return count;
23     }
24
25     int size_set(int i)
26     {
27         return sz[find_set(i)];
28     }
29
30     int find_set(int i)
31     {
32         return (id[i] == i) ? i : (id[i] = find_set(id[i]));
33     }
34
35     bool connected(int i, int j)
36     {
37         return find_set(i) == find_set(j);
38     }
39

```



```

40     void union_set(int p, int q)
41     {
42         int i = find_set(p);
43         int j = find_set(q);
44         if (i == j) return;
45         // make smaller root point to larger one
46         if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
47         else { id[j] = i; sz[i] += sz[j]; }
48         count--;
49     }
50 };
51
52 using namespace std;
53
54 const int MAX_N = 10010;
55
56 int main(int argc, char *argv[])
57 {
58     int TC, N, M;
59     int debts[MAX_N];
60     int balance[MAX_N];
61     scanf("%d", &TC);
62     while (TC--)
63     {
64         scanf("%d_%d", &N, &M);
65         union_find uf(N);
66         for (int i = 0; i < N; ++i)
67         {
68             scanf("%d", &debts[i]);
69         }
70         while (M--)
71         {
72             int A, B;
73             scanf("%d_%d", &A, &B);
74             uf.union_set(A, B);
75         }
76         memset(balance, 0, sizeof balance);
77         for (int i = 0; i < N; ++i)
78         {
79             balance[uf.find_set(i)] += debts[i];
80         }
81
82         if (all_of(balance, balance + N, [](int v){ return v == 0; })) printf("POSSIBLE\n");
83         else printf("IMPOSSIBLE\n");
84     }
85     return 0;
86 }

```

14 UVA 12086: Potentiometers

```

1 class segment_tree
2 {
3 private:
4     std::vector<int> st, A;

```

```

5  int N;
6  inline int left (int p) { return p << 1; }
7  inline int right(int p) { return (p << 1) + 1; }
8
9  void build(int p, int L, int R)
10 {
11     if (L == R)
12         st[p] = A[L];
13     else
14     {
15         build(left(p) , L , (L + R) / 2);
16         build(right(p), (L + R) / 2 + 1, R );
17         st[p] = st[left(p)] + st[right(p)];
18     }
19 }
20
21 int find(int p, int L, int R, int i, int j)
22 {
23     if (i > R || j < L) return 0;
24     if (L >= i && R <= j) return st[p];
25
26     int sum1 = find(left(p) , L , (L+R) / 2, i, j);
27     int sum2 = find(right(p), (L+R) / 2 + 1, R , i, j);
28
29     return sum1 + sum2;
30 }
31
32 int update_point(int p, int L, int R, int idx, int new_value)
33 {
34     if (idx > R || idx < L)
35         return st[p];
36
37     if (L == idx && R == idx)
38     {
39         A[idx] = new_value;
40         return st[p] = new_value;
41     }
42
43     int sum1 = update_point(left(p) , L , (L + R) / 2, idx, new_value);
44     int sum2 = update_point(right(p), (L + R) / 2 + 1, R , idx, new_value);
45
46     return st[p] = sum1 + sum2;
47 }
48
49 int update_range(int p, int L, int R, int i, int j, int new_value)
50 {
51     if (i > R || j < L) return 0;
52
53     if (L == R)
54     {
55         A[L] = new_value;
56         return st[p] = new_value;
57     }
58
59     int sum1 = update_range(left(p) , L , (L + R) / 2, i, j, new_value);
60     int sum2 = update_range(right(p), (L + R) / 2 + 1, R , i, j, new_value);
61

```

```

62     return st[p] = sum1 + sum2;
63 }
64
65 public:
66     segment_tree(const std::vector<int>& _A)
67     {
68         A = _A; N = (int)A.size();
69         st.assign(4 * N, 0);
70         build(1, 1, N - 1);
71     }
72
73     int find(int i, int j)
74     {
75         return find(1, 1, N - 1, i, j);
76     }
77
78     int update_point(int idx, int new_value)
79     {
80         return update_point(1, 1, N - 1, idx, new_value);
81     }
82
83     int update_range(int i, int j, int new_value)
84     {
85         return update_range(1, 1, N - 1, i, j, new_value);
86     }
87 };
88
89 int main(int argc, char *argv[])
90 {
91     int N;
92     int t = 0;
93     while (scanf("%d", &N), t++, N) {
94         if (t > 1) printf("\n");
95         vector<int> resistances(N+1, 0);
96         for (int i = 1; i <= N; ++i)
97             {
98                 scanf("%d", &resistances[i]);
99             }
100         segment_tree st(resistances);
101         char cmd[4];
102         printf("Case_%d:\n", t);
103         while (scanf("%s", cmd), strcmp(cmd, "END")) {
104             int a, b;
105             scanf("%d_%d", &a, &b);
106             if (cmd[0] == 'S') {
107                 st.update_point(a, b);
108             } else {
109                 printf("%d\n", st.find(a, b));
110             }
111         }
112     }
113     return 0;
114 }

```

15 UVA 10909: Lucky Number

```
1  const int N = 2000010;
2  bitset<N> lucky_numbers;
3
4  class segment_tree
5  {
6  private:
7      std::vector<int> st;
8      inline int left (int p) { return p << 1; }
9      inline int right(int p) { return (p << 1) + 1; }
10
11     void build(int p, int L, int R)
12     {
13         if (L == R)
14             st[p] = lucky_numbers[L];
15         else
16         {
17             build(left(p) , L, (L + R) / 2);
18             build(right(p), (L + R) / 2 + 1, R);
19             int n1 = st[left(p)], n2 = st[right(p)];
20             st[p] = n1 + n2;
21         }
22     }
23
24     int find(int p, int L, int R, int k)
25     {
26         if (L >= N) return -1;
27         if (L == R) return L;
28         int n1 = st[left(p)];
29         if (n1 >= k) return find(left(p) , L, (L+R) / 2, k);
30         return find(right(p), (L+R) / 2 + 1, R, k - n1);
31     }
32
33     void clear_point(int p, int L, int R, int k)
34     {
35         if (L == R)
36         {
37             lucky_numbers.reset(L);
38             st[p] = 0;
39             return;
40         }
41         int n1 = st[left(p)];
42         if (n1 >= k)
43         {
44             clear_point(left(p), L, (L + R) / 2, k);
45             n1 = st[left(p)];
46         }
47         else
48             clear_point(right(p), (L + R) / 2 + 1, R, k - n1);
49         int n2 = st[right(p)];
50         st[p] = n1 + n2;
51     }
52
53 public:
54     segment_tree()
```

```

55     {
56         st.assign(4 * N, 0);
57     }
58
59     void init()
60     {
61         build(1, 0, N - 1);
62     }
63
64     int find(int k)
65     {
66         return find(1, 0, N, k);
67     }
68
69     void clear_point(int k)
70     {
71         clear_point(1, 0, N - 1, k);
72     }
73 };
74
75 void init(segment_tree& st)
76 {
77     lucky_numbers.set();
78     for (int i = 0; i < N; i += 2)
79     {
80         lucky_numbers.reset(i);
81     }
82     st.init();
83     int n = N / 2 + 1;
84     int k = 2;
85     while (true)
86     {
87         int l = st.find(k++);
88         if (l > n) break;
89         int j = 0;
90         for (int i = l; i <= n; i += 1)
91         {
92             st.clear_point(i - j);
93             ++j;
94         }
95         n -= j;
96     }
97 }
98
99 int main(int argc, char *argv[])
100 {
101     segment_tree st;
102     init(st);
103     int N;
104     while (scanf("%d", &N) != EOF)
105     {
106         int L1 = -1, L2 = -1;
107         if ((N & 1) == 0)
108         {
109             int i = N/2;
110             for (; i > 0 && !lucky_numbers[i]; --i);
111             for (; i > 0; i -= 2)

```

```

112     {
113         if (lucky_numbers[i] && lucky_numbers[N - i])
114         {
115             L1 = i;
116             L2 = N - i;
117             break;
118         }
119     }
120 }
121 if (L1 == -1) printf("%d is not the sum of two luckies!\n", N);
122 else printf("%d is the sum of %d and %d.\n", N, L1, L2);
123 }
124 return 0;
125 }

```

16 UVA 750: 8 Queens Chess Problem

```

1  #include <cstdio>
2  #include <vector>
3  #include <cmath>
4
5  template <typename T>
6  struct domain
7  {
8      std::vector<T> candidate;
9
10     domain(int size) : candidate(size + 1)
11     {
12     }
13     virtual bool is_a_solution(int k) = 0;
14     virtual void process_solution(int k) = 0;
15     virtual void next(std::vector<T>& possibilities, int k) = 0;
16     virtual void make_move(int k)
17     {
18     }
19     virtual void unmake_move(int k)
20     {
21     }
22     virtual bool stop()
23     {
24         return false;
25     }
26     void set(int i, const T& elem)
27     {
28         candidate[i] = elem;
29     }
30 };
31
32 template <typename T>
33 class backtracking
34 {
35     domain<T>& dom;
36 public:
37     backtracking(domain<T>& dom) : dom(dom)
38     {

```

```

39     }
40
41     void backtrack(int k)
42     {
43         if (dom.is_a_solution(k))
44         {
45             dom.process_solution(k);
46         }
47         else
48         {
49             k++;
50             std::vector<T> possibilities;
51             dom.next(possibilities, k);
52             for (int i = 0; i < possibilities.size(); ++i)
53             {
54                 dom.set(k, possibilities[i]);
55                 dom.make_move(k);
56                 backtrack(k);
57                 dom.unmake_move(k);
58                 if (dom.stop()) return;
59             }
60         }
61     }
62
63     void execute()
64     {
65         backtrack(0);
66     }
67 };
68
69 template<typename T>
70 backtracking<T> make_backtracking(domain<T>& dom)
71 {
72     return backtracking<T>(dom);
73 }
74
75 using namespace std;
76
77 int a, b;
78
79 class height_queens : public domain<int>
80 {
81 private:
82     int size;
83     int solution_count = 1;
84 public:
85     height_queens(int size) : domain(size)
86     {
87         this->size = size;
88     }
89     bool is_a_solution(int k)
90     {
91         return (k == size && candidate[b] == a);
92     }
93     void process_solution(int k)
94     {
95         printf("%2d\UUUUUU%d", solution_count++, candidate[1]);

```

```

96     for (int i = 2; i <= size; i++)
97     {
98         printf("%d", candidate[i]);
99     }
100    printf("\n");
101 }
102 void next(std::vector<int>& possibilities, int k)
103 {
104     for (int i = 1; i <= size; i++)
105     {
106         bool legal_move = true;
107         for (int j = 1; j < k; j++)
108         {
109             if (abs(k-j) == abs(i-candidate[j]) // diagonal threat
110                 || i == candidate[j]) // line threat
111             {
112                 legal_move = false;
113                 break;
114             }
115         }
116         if (legal_move)
117         {
118             possibilities.push_back(i);
119         }
120     }
121 }
122 };
123
124 int main(int argc, char *argv[])
125 {
126     int TC;
127     scanf("%d", &TC);
128     while (TC--)
129     {
130         scanf("%d_%d", &a, &b);
131         printf("SOLN_0000000000COLUMN\n");
132         printf("#_000000001_2_3_4_5_6_7_8\n\n");
133         height_queens queens(8);
134         auto bt = make_backtracking(queens);
135         bt.execute();
136         if (TC) printf("\n");
137     }
138     return 0;
139 }

```

17 UVA 357: Let Me Count the Ways

```

1  #include <cstdio>
2  #include <cstring>
3
4  using namespace std;
5
6  #define NB_COINS 5
7  #define MAX_MONEY 30000
8

```



```

9 long long ways[NB_COINS + 1][MAX_MONEY + 1];
10 int coins[NB_COINS + 1] = {0, 1, 5, 10, 25, 50};
11 int N;
12
13 long long solve(int index, int sum)
14 {
15     if (sum < 0) return 0;
16     if (index == NB_COINS + 1) return sum == 0;
17     if (ways[index][sum] != -1) return ways[index][sum];
18     long long res = solve(index, sum - coins[index]) + solve(index + 1, sum);
19     return ways[index][sum] = res;
20 }
21
22 int main(int argc, char *argv[])
23 {
24     memset(ways, -1, sizeof(ways));
25     while (scanf("%d", &N) != EOF)
26     {
27         solve(1, N);
28         if (ways[1][N] == 1) printf("There is only 1 way to produce %d cents change.\n",
N);
29         else printf("There are %lld ways to produce %d cents change.\n", ways[1][N], N);
30     }
31     return 0;
32 }

```

```

1 #include <stdio>
2 #include <cstring>
3
4 using namespace std;
5
6 #define NB_COINS 5
7 #define MAX_MONEY 30000
8
9 int main(int argc, char *argv[])
10 {
11     int N;
12     long ways[NB_COINS + 1][MAX_MONEY + 1];
13     int coins[NB_COINS + 1] = {0, 1, 5, 10, 25, 50};
14     memset(ways, 0, sizeof ways);
15     ways[0][0] = 1;
16     for (int i = 1; i <= NB_COINS; ++i)
17     {
18         ways[i][0] = 1;
19         for (int j = 1; j <= MAX_MONEY; ++j)
20         {
21             long n = 0;
22             if (j - coins[i] >= 0) n = ways[i][j - coins[i]];
23             n += ways[i - 1][j];
24             ways[i][j] = n;
25         }
26     }
27     while (scanf("%d", &N) != EOF)
28     {
29         if (ways[5][N] == 1) printf("There is only 1 way to produce %d cents change.\n", N)
;

```

```

30     else printf("There are %ld ways to produce %d cents change.\n", ways[5][N], N);
31 }
32 return 0;
33 }

```

18 UVA 10131: Is Bigger Smarter?

```

1  int main(int argc, char *argv[])
2  {
3      vector<tuple<int, int, int>> elephants; //weight, IQ, index
4      int lis[1001];
5      int prev[1001];
6      memset(prev, -1, sizeof prev);
7      int W, IQ;
8      int cpt = 0;
9      while (scanf("%d %d", &W, &IQ) != EOF)
10     {
11         elephants.push_back(make_tuple(W, IQ, ++cpt));
12     }
13     sort(elephants.begin(), elephants.end());
14
15     lis[0] = 1;
16     int ans = 1, index = 0;
17     for (int i = 1; i < elephants.size(); ++i)
18     {
19         int best_index = -1;
20         lis[i] = 1;
21         for (int j = 0; j < i; ++j)
22         {
23             if (get<0>(elephants[j]) != get<0>(elephants[i]) && get<1>(elephants[j]) > get
<1>(elephants[i]) && 1 + lis[j] > lis[i])
24             {
25                 lis[i] = 1 + lis[j];
26                 best_index = j;
27             }
28         }
29         prev[i] = best_index;
30         if (lis[i] > ans)
31         {
32             ans = lis[i];
33             index = i;
34         }
35     }
36
37     stack<int> s;
38     s.push(get<2>(elephants[index]));
39     while (prev[index] != -1)
40     {
41         index = prev[index];
42         s.push(get<2>(elephants[index]));
43     }
44     printf("%d\n", ans);
45     while (!s.empty())
46     {
47         printf("%d\n", s.top());

```

```
48     s.pop();
49     }
50     return 0;
51 }
```

19 UVA 10536: Game of Euler

```
1 unsigned short board = 0;
2 //[15][14][13][12]
3 //[11][10][ 9][ 8]
4 //[ 7][ 6][ 5][ 4]
5 //[ 3][ 2][ 1][ 0]
6
7 unsigned short moves[] =
8 {
9     0x8000,
10    0x4000,
11    0x2000,
12    0x1000,
13    0x0800,
14    0x0400,
15    0x0200,
16    0x0100,
17    0x0080,
18    0x0040,
19    0x0020,
20    0x0010,
21    0x0008,
22    0x0004,
23    0x0002,
24    0x0001,
25
26    0xC000,
27    0x3000,
28    0x0C00,
29    0x0300,
30    0x00C0,
31    0x0030,
32    0x000C,
33    0x0003,
34
35    0x8800,
36    0x4400,
37    0x2200,
38    0x1100,
39    0x0088,
40    0x0044,
41    0x0022,
42    0x0011,
43
44    0xE000,
45    0x7000,
46    0x0E00,
47    0x0700,
48    0x00E0,
```

```

49     0x0070,
50     0x000E,
51     0x0007,
52
53     0x8880,
54     0x4440,
55     0x2220,
56     0x1110,
57     0x0888,
58     0x0444,
59     0x0222,
60     0x0111
61 };
62
63 const int NB_MOVES = 48;
64 char memo[1<<16][2];
65
66 bool is_terminal(unsigned short board)
67 {
68     return board == 0xFFFF;
69 }
70
71 bool move_possible(int move, unsigned short board)
72 {
73     return !(board & moves[move]);
74 }
75
76 int min_value(unsigned short board);
77 int max_value(unsigned short board)
78 {
79     if (is_terminal(board)) return 1;
80     if (memo[board][0] != 0) return memo[board][0];
81     int v = -2;
82     for (int i = 0; i < NB_MOVES; ++i)
83     {
84         if (move_possible(i, board))
85         {
86             v = max(v, min_value(board | moves[i]));
87         }
88     }
89     return memo[board][0] = v;
90 }
91
92 int min_value(unsigned short board)
93 {
94     if (is_terminal(board)) return -1;
95     if (memo[board][1] != 0) return memo[board][1];
96     int v = 2;
97     for (int i = 0; i < NB_MOVES; ++i)
98     {
99         if (move_possible(i, board))
100         {
101             v = min(v, max_value(board | moves[i]));
102         }
103     }
104     return memo[board][1] = v;
105 }

```

```

106
107 int main(int argc, char *argv[])
108 {
109     int TC;
110     scanf("%d", &TC);
111     memset(memo, 0, sizeof memo);
112     while (TC--)
113     {
114         board = 0;
115         for (int i = 15; i >= 0; --i)
116         {
117             char c;
118             scanf("%c", &c);
119             board |= (c == 'X') << i;
120         }
121         if (max_value(board) == 1) printf("WINNING\n");
122         else printf("LOSING\n");
123     }
124     return 0;
125 }

```

20 UVA 1213: Sum of Different Primes

```

1  const int MAX_N = 1121;
2  const int MAX_K = 15;
3
4  int N, K;
5  int primes[] =
6  {
7      2,      3,      5,      7,      11,      13,      17,      19,      23,      29,
8      31,      37,      41,      43,      47,      53,      59,      61,      67,      71,
9      73,      79,      83,      89,      97,      101,      103,      107,      109,      113,
10     127,      131,      137,      139,      149,      151,      157,      163,      167,      173,
11     179,      181,      191,      193,      197,      199,      211,      223,      227,      229,
12     233,      239,      241,      251,      257,      263,      269,      271,      277,      281,
13     283,      293,      307,      311,      313,      317,      331,      337,      347,      349,
14     353,      359,      367,      373,      379,      383,      389,      397,      401,      409,
15     419,      421,      431,      433,      439,      443,      449,      457,      461,      463,
16     467,      479,      487,      491,      499,      503,      509,      521,      523,      541,
17     547,      557,      563,      569,      571,      577,      587,      593,      599,      601,
18     607,      613,      617,      619,      631,      641,      643,      647,      653,      659,
19     661,      673,      677,      683,      691,      701,      709,      719,      727,      733,
20     739,      743,      751,      757,      761,      769,      773,      787,      797,      809,
21     811,      821,      823,      827,      829,      839,      853,      857,      859,      863,
22     877,      881,      883,      887,      907,      911,      919,      929,      937,      941,
23     947,      953,      967,      971,      977,      983,      991,      997,      1009,      1013,
24     1019,      1021,      1031,      1033,      1039,      1049,      1051,      1061,      1063,      1069,
25     1087,      1091,      1093,      1097,      1103,      1109,      1117,      1123
26 };
27
28 const int NB_PRIMES = sizeof(primes) / sizeof(int);
29
30 int memo[NB_PRIMES][MAX_K][MAX_N];
31
32 int nb_sums(int index, int nb_remaining, int sum)

```

```

33 {
34     if (nb_remaining == 0) return sum == 0;
35     if (index == NB_PRIMES) return 0;
36     if (memo[index][nb_remaining][sum] != -1) return memo[index][nb_remaining][sum];
37     return memo[index][nb_remaining][sum] =
38         (sum - primes[index] < 0 ? 0 : nb_sums(index + 1, nb_remaining - 1, (sum - primes[
index])))
39         + nb_sums(index + 1, nb_remaining, sum);
40 }
41
42 int main(int argc, char *argv[])
43 {
44     memset(memo, -1, sizeof memo);
45     while (scanf("%d%d", &N, &K), (N || K))
46     {
47         printf("%d\n", nb_sums(0, K, N));
48     }
49     return 0;
50 }

```

21 UVA 11517: Exact Change

```

1  const int MAX_MONEY = 10010;
2
3  int main(int argc, char *argv[])
4  {
5      int change[MAX_MONEY];
6      int TC;
7      scanf("%d", &TC);
8      while (TC--)
9      {
10         int V, N;
11         scanf("%d%d", &V, &N);
12         fill(change, change + MAX_MONEY, INT_MAX);
13         change[0] = 0;
14         for (int i = 0; i < N; ++i)
15         {
16             int coin;
17             scanf("%d", &coin);
18             for (int j = MAX_MONEY - 1; j >= coin; --j)
19             {
20                 if (change[j - coin] != INT_MAX) change[j] = min(change[j], 1 + change[j -
coin]);
21             }
22         }
23         for (int i = V; i < MAX_MONEY; ++i)
24         {
25             if (change[i] != INT_MAX)
26             {
27                 printf("%d%d\n", i, change[i]);
28                 break;
29             }
30         }
31     }
32     return 0;

```

22 Hackerrank Dynamic Programming: Bricks Game

```

1  #include <algorithm>
2  #include <vector>
3  #include <iostream>
4  using namespace std;
5
6  int main(int argc, char *argv[])
7  {
8      int TC;
9      cin >> TC;
10     while (TC--)
11     {
12         int N;
13         cin >> N;
14         vector<long long> bricks(N + 1, 0);
15         for (int i = N; i >= 1; --i)
16         {
17             cin >> bricks[i];
18         }
19         vector<long long> sum(N + 1, 0);
20         for (int i = 1; i <= N; ++i)
21         {
22             sum[i] += sum[i - 1] + bricks[i];
23         }
24         vector<long long> dp(N + 1, 0);
25         dp[1] = sum[1];
26         dp[2] = sum[2];
27         dp[3] = sum[3];
28         for (int i = 4; i <= N; ++i)
29         {
30             dp[i] = max(bricks[i] + sum[i - 1] - dp[i - 1],
31                        max(bricks[i] + bricks[i - 1] + sum[i - 2] - dp[i - 2],
32                           bricks[i] + bricks[i - 1] + bricks[i - 2] + sum[i - 3] - dp[i -
33                3]));
34         }
35         cout << dp[N] << endl;
36     }
37     return 0;

```

Index

8 Queens Chess Problem, 21

Add Bricks in the Wall, 4

Almost Union Find, 13

Anagram, 1

Argus, 6

Bricks Game, 30

Counterfeit Dollar, 1

Exact Change, 29

Exact Sum, 3

Fable Rooks, 7

Game of Euler, 26

Grey Codes, 5

Is Bigger Smarter?, 25

Jolly Jumpers, 3

Let Me Count the Ways, 23

Lucky Number, 19

Money Matters, 15

Network Connections, 10

Potentiometers, 16

Sum of Different Primes, 28

Ubiquitous Religions, 12

UVA 10038, 3

UVA 10131, 25

UVA 10507, 8

UVA 10536, 26

UVA 10583, 12

UVA 10909, 19

UVA 11040, 4

UVA 11057, 3

UVA 11134, 7

UVA 11173, 5

UVA 11517, 29

UVA 11690, 15

UVA 11987, 13

UVA 1203, 6

UVA 12086, 16

UVA 1213, 28

UVA 195, 1

UVA 357, 23

UVA 608, 1

UVA 750, 21

UVA 793, 10

Waking Up Brain, 8