Competitive Programming Solution Manual

Pascal Garcia

February 4, 2016

1 UVA 195: Anagram

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cctype>

using namespace std;

bool comparator(char a, char b)
{
  if (tolower(a) == tolower(b))
  {
    return a<b;
  }
  return tolower(a) < tolower(b);
}

int main()
{
  char word[1000];
  int n;

  scanf("%d", &n);
  while (n--)
  {
    scanf("␣%s", word);

    sort(word, word+strlen(word), comparator);
    printf("%s\n", word);
    while (next_permutation(word, word+strlen(word), comparator))
    {
      printf("%s\n", word);
    }
  }
  return 0;
}
```

2 UVA 608: Counterfeit Dollar

```
#include <cstdio>
#include <cstring>
#include <climits>
#include <cstdlib>

char plateau1[13];
char plateau2[13];
char scale[13];
int coins[12];

#define DOLLAR INT_MAX

void update_up(char plateau[13], int i)
{
```

```c
  int index = plateau[i] - 'A';
  if (coins[index] != DOLLAR) coins[index]--;
}

void update_down(char plateau[13], int i)
{
  int index = plateau[i] - 'A';
  if (coins[index] != DOLLAR) coins[index]++;
}

void deduction()
{
  int i = 0;
  while (plateau1[i])
  {
    if (!strcmp(scale, "even"))
    {
      coins[plateau1[i] - 'A'] = DOLLAR;
      coins[plateau2[i] - 'A'] = DOLLAR;
    }
    else if (!strcmp(scale, "up"))
    {
      update_up(plateau2, i);
      update_down(plateau1, i);
    }
    else if (!strcmp(scale, "down"))
    {
      update_down(plateau2, i);
      update_up(plateau1, i);
    }
    i++;
  }
}

int main() {
  int T;
  scanf("%d", &T);
  while (T--)
  {
    memset(coins, 0, sizeof coins);
    for (int i=0; i < 3; ++i)
    {
      scanf(" %s %s %s", plateau1, plateau2, scale);
      deduction();
    }
    int index = -1, max = 0;
    for (int i = 0; i < 12; i++)
    {
      if (coins[i] != DOLLAR && abs(coins[i]) >= max)
      {
        index = i;
        max = abs(coins[i]);
      }
    }
    printf("%c is the counterfeit coin and it is %s\n", 'A' + index, (coins[index] < 0
? "light." : "heavy."));
  }
  return 0;
}
```

3 UVA 11057: Exact Sum

```
#include <cstdio>
#include <algorithm>
#include <vector>
#include <climits>

using namespace std;

int main(int argc, char *argv[])
{
  vector<int> prices(10001);
  int nb_books;
  while (scanf("%d", &nb_books) != EOF)
  {
    prices.clear();
    while (nb_books--)
    {
      int v;
      scanf("%d", &v);
      prices.push_back(v);
    }
    int money;
    scanf("%d", &money);
    sort(prices.begin(), prices.end());
    int price1, price2;
    int best = INT_MAX;
    for (vector<int>::iterator it = prices.begin(); it != prices.end(); it++)
    {
      int target = money - *it;
      if (target < 0) break;
      if (binary_search(it + 1, prices.end(), target))
      {
        int delta = abs(target - *it);
        if (delta < best)
        {
          best = delta;
          price1 = min(*it, target);
          price2 = max(*it, target);
        }
      }
    }
    printf("Peter␣should␣buy␣books␣whose␣prices␣are␣%d␣and␣%d.\n\n", price1, price2);
  }
  return 0;
}
```

4 UVA 10038: Jolly Jumpers

```
#include <cstdio>
```

```
#include <cstdlib>
#include <vector>

using namespace std;

int main(int argc, char *argv[])
{
  int n;
  while (scanf("%d", &n) != EOF)
  {
    vector<bool> sequence(n, false);
    int prev, current;
    scanf("%d", &prev);
    int remaining = n-1;
    int cpt = 0;
    while (remaining--)
    {
      scanf("%d", &current);
      int v = abs(prev - current);
      if (v < n && v != 0 && !sequence[v])
      {
        sequence[v] = true;
        cpt++;
      }
      prev = current;
    }
    if (cpt == n - 1)
    {
      printf("Jolly\n");
    }
    else
    {
      printf("Not␣jolly\n");
    }
  }
  return 0;
}
```

5 UVA 11040: Add Bricks in the Wall

```
#include <cstdio>
#include <cstring>

using namespace std;

int bricks[9][9];

void fill()
{
  for (int i = 0; i < 7; i += 2)
  {
    bricks[0][i+1] = (bricks[2][i] - bricks[0][i] - bricks[0][i+2]) / 2;
  }
  for (int i = 0; i < 8; ++i)
  {
    for (int j = 0; j < 8; ++j)
    {
      bricks[i+1][j] = bricks[i][j] + bricks[i][j+1];
    }
  }
}

void display()
{
  for (int i = 8; i >= 0; --i)
  {
    printf("%d", bricks[i][0]);
    for (int j = 1; j < 9 - i; ++j)
    {
      printf("␣%d", bricks[i][j]);
    }
    printf("\n");
  }
}

int main(int argc, char *argv[])
{
  int TC;
  scanf("%d", &TC);
  while (TC--)
  {
    int cpt = 0;
    memset(bricks, 0, sizeof bricks);
    while (cpt < 6)
    {
      scanf("%*d");
      cpt++;
    }
    int i = 0;
    while (cpt < 10)
    {
      scanf("%d", &bricks[2][i]);
      i+= 2;
      cpt++;
    }
    i = 0;
    while (cpt++ < 15)
    {
      scanf("%d", &bricks[0][i]);
      i+= 2;
    }
    fill();
    display();
  }
  return 0;
}
```

6 UVA 11173: Grey Codes

```
#include <cstdio>
```

```

using namespace std;

int main(int argc, char *argv[])
{
  int TC;
  scanf("%d", &TC);
  while (TC--)
  {
    unsigned int n, k;
    scanf("%d␣%d", &n, &k);
    printf("%d\n", k ^ (k >> 1));
  }
  return 0;
}
```

7 UVA 1203: Argus

```
include <cstdio>
include <vector>
include <queue>

using namespace std;

class comparison
{
  public:
  bool operator() (const pair<int, int>& p1, const pair<int, int>& p2) const
  {
    return p1.first > p2.first || (p1.first == p2.first && p1.second > p2.second);
  }
};

int main(int argc, char *argv[])
{
  char s[1024];
  vector<int> id_to_period(3001);
  priority_queue<pair<int, int>, vector<pair<int, int>>, comparison> pq;
  while (scanf("␣%s", s), s[0] != '#')
  {
    int id, period;
    scanf("%d␣%d", &id, &period);
    id_to_period[id] = period;
    pq.push(make_pair(period, id));
  }
  int K;
  scanf("%d", &K);
  while (K--)
  {
    pair<int, int> item = pq.top();
    pq.pop();
    printf("%d\n", item.second);
    pq.push(make_pair(item.first + id_to_period[item.second], item.second));
  }
  return 0;
}
```

8 UVA 11134: Fable Rooks

```
struct range
{
  int left, right, id;
  bool operator<(const range& range) const
  {
    return right > range.right;
  }
};

const int MAX_N = 5010;

bool solve(int N, vector<range> ranges[MAX_N], int coord[MAX_N])
{
  priority_queue<range> pq;
  for (int i = 1; i <= N; ++i)
    {
      for_each(ranges[i].begin(), ranges[i].end(), [&pq](const range& r){ pq.push(r);
});
      if (pq.size() == 0) return false;
      range r = pq.top();
      pq.pop();
      if (r.right < i) return false;
      coord[r.id] = i;
    }
  return true;
}

int main(int argc, char *argv[])
{
  int N;
  vector<range> xrange[MAX_N];
  vector<range> yrange[MAX_N];
  int rook_x[MAX_N];
  int rook_y[MAX_N];
  while (scanf("%d", &N), N)
    {
      range x;
      range y;
      for (int i = 1; i <= N; ++i)
      {
        xrange[i].clear();
        yrange[i].clear();
      }
      for (int i = 0; i < N; ++i)
      {
        scanf("%d%d%d%d", &x.left, &y.left, &x.right, &y.right);
        x.id = i;
        y.id = i;
        xrange[x.left].push_back(x);
        yrange[y.left].push_back(y);
      }
      bool possible = solve(N, xrange, rook_x) && solve(N, yrange, rook_y);
      if (possible)
      {
        for (int i = 0; i < N; ++i)
        {
          printf("%d␣%d\n", rook_x[i], rook_y[i]);
        }
      }
      else printf("IMPOSSIBLE\n");
    }
  return 0;
}
```

9 UVA 10507: Waking Up Brain

```
#include <cstdio>
#include <vector>
#include <string>
#include <sstream>
#include <queue>

class graph
{
  public:
  typedef std::vector<std::pair<int, float>> NEIGHBORS;
  typedef std::vector<NEIGHBORS> ADJ_LIST;

  private:
  const int nb_vertices;
  int nb_edges;
  ADJ_LIST adj;
  bool directed;

  public:
  graph(int n = 1000, bool directed = false) : nb_vertices(n), nb_edges(0), directed(
directed)
  {
    adj.assign(n, NEIGHBORS());
  }

  int vertices()
  {
    return nb_vertices;
  }

  int edges()
  {
    return nb_edges;
  }

  int degree(int i)
  {
    return adj[i].size();
  }

  void add_edge(int i, int j, float weight = 1)
  {
    nb_edges++;
    adj[i].push_back(std::make_pair(j, weight));
    if (!directed) adj[j].push_back(std::make_pair(i, weight));
  }

  const NEIGHBORS& neighbors(int i)
  {
    return adj[i];
  }

  bool connected(int i, int j)
  {
    for (const auto& edge : adj[i])
    {
      if (edge.first == j) return true;
    }
    return false;
  }

  std::string to_string()
  {
    std::stringstream res;
    res << nb_vertices << "␣vertices,␣" << nb_edges << "␣edges\n";
    for (int i = 0; i < nb_vertices; i++)
    {
      res << i << ":␣";
      for (const auto& edge : adj[i])
      {
        res << "(" << edge.first << ",␣w:" << edge.second << ")␣";
      }
      res << "\n";
    }
    return res.str();
  }
};

using namespace std;

#define CHAR_TO_ZONE(C) ((C) - 'A')

int main()
{
  for (int slept_areas, connections; scanf("%d␣%d", &slept_areas, &connections) == 2;)
  {
    char z1, z2, z3;
    int activated = 0;
    vector<int> zone_nb_activation(26, 0);
    queue<pair<int, int>> zone_activation_time;

    scanf("␣%c%c%c", &z1, &z2, &z3);

    zone_activation_time.push(make_pair(CHAR_TO_ZONE(z1), 0));
    zone_nb_activation[CHAR_TO_ZONE(z1)] = 3;
    zone_activation_time.push(make_pair(CHAR_TO_ZONE(z2), 0));
    zone_nb_activation[CHAR_TO_ZONE(z2)] = 3;
    zone_activation_time.push(make_pair(CHAR_TO_ZONE(z3), 0));
    zone_nb_activation[CHAR_TO_ZONE(z3)] = 3;

    graph g(26);

    while (connections--)
    {
      scanf("␣%c%c", &z1, &z2);
      g.add_edge(CHAR_TO_ZONE(z1), CHAR_TO_ZONE(z2));
    }

    int time = 0;
    while (!zone_activation_time.empty())
    {
      auto& zone = zone_activation_time.front();
      time = zone.second;
      zone_activation_time.pop();
      activated++;
      for (auto& p : g.neighbors(zone.first))
      {
        if (++zone_nb_activation[p.first] == 3) zone_activation_time.push(make_pair(p.
first, zone.second + 1));
      }
    }
    if (activated == slept_areas) printf("WAKE␣UP␣IN,␣%d,␣YEARS\n", time);
    else printf("THIS␣BRAIN␣NEVER␣WAKES␣UP\n");
  }
  return 0;
}
```

10 UVA 793: Network Connections

```
#include <cstdio>
#include <vector>

class union_find
{
  private:
  std::vector<int> id; // id[i] = parent of i
  std::vector<int> sz; // sz[i] = number of objects in subtree rooted at i
  int count; // number of components
  public:
  union_find(int N)
  {
    count = N;
    id.assign(N, 1);
    sz.assign(N, 0);
    for (int i = 0; i < N; i++)
    {
      id[i] = i;
      sz[i] = 1;
    }
  }

  int nb_components()
```

```cpp
  {
    return count;
  }

  int size_set(int i)
  {
    return sz[find_set(i)];
  }

  int find_set(int i)
  {
    return (id[i] == i) ? i : (id[i] = find_set(id[i]));
  }

  bool connected(int i, int j)
  {
    return find_set(i) == find_set(j);
  }

  void union_set(int p, int q)
  {
    int i = find_set(p);
    int j = find_set(q);
    if (i == j) return;
    // make smaller root point to larger one
    if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
    else               { id[j] = i; sz[i] += sz[j]; }
    count--;
  }
};

using namespace std;

int main()
{
  int TC;
  scanf("%d", &TC);
  while (TC--)
  {
    int N;
    scanf("%d%*c", &N);
    union_find uf(N+1);
    int total = 0;
    int successful = 0;
    char c = getchar();
    while (c != EOF && c != '\n')
    {
      int a, b;
      scanf("%d␣%d%*c", &a, &b);
      if (c == 'c')
      {
        uf.union_set(a, b);
      }
      else
      {
        total++;
        if (uf.connected(a, b))
        {
          successful++;
        }
      }
      c = getchar();
    }
    printf("%d,%d\n", successful, total - successful);
    if (TC) printf("\n");
  }
  return 0;
}
```

11 UVA 10583: Ubiquitous Religions

```
class union_find
{
private:
  std::vector<int> id; // id[i] = parent of i
  std::vector<int> sz; // sz[i] = number of objects in subtree rooted at i
  int count; // number of components
public:
  union_find(int N)
    {
      count = N;
      id.assign(N, 1);
      sz.assign(N, 0);
      for (int i = 0; i < N; i++)
      {
        id[i] = i;
        sz[i] = 1;
      }
    }

    int nb_components()
    {
      return count;
    }

    int size_set(int i)
    {
      return sz[find_set(i)];
    }

    int find_set(int i)
    {
      return (id[i] == i) ? i : (id[i] = find_set(id[i]));
    }

    bool connected(int i, int j)
    {
      return find_set(i) == find_set(j);
    }

    void union_set(int p, int q)
    {
      int i = find_set(p);
      int j = find_set(q);
      if (i == j) return;
      // make smaller root point to larger one
      if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
      else               { id[j] = i; sz[i] += sz[j]; }
      count--;
    }
};

using namespace std;

int main(int argc, char *argv[])
{
  int N, M, cpt = 1;
  while (scanf("%d␣%d", &N, &M), (N || M))
    {
      union_find uf(N);
      while (M--)
      {
        int a, b;
        scanf("%d␣%d", &a, &b);
        uf.union_set(a-1, b-1);
      }
      printf("Case␣%d:␣%d\n", cpt++, uf.nb_components());
    }
  return 0;
}
```

12 UVA 11987: Almost Union Find

```
#include <cstdio>
#include <vector>

class union_find
{
private:
  std::vector<int> id; // id[i] = parent of i
  std::vector<int> sz; // sz[i] = number of objects in subtree rooted at i
  std::vector<long> sum; // sum[i] = sum of elements in subtree rooted at i
  int count; // number of components
  int N;
public:
  union_find(int N)
  {
    this->N = N;
    count = N;
    id.assign(2 * N + 1, 0);
    sz.assign(N + 1, 1);
    sum.assign(N + 1, 0);
    for (int i = 1; i <= N; i++)
      {
        id[i] = id[N + i] = N + i;
        sum[i] = i;
      }
  }

  int nb_components()
  {
    return count;
  }

  int size_set(int i)
  {
    return sz[find_set(i) - N];
  }

  int sum_set(int i)
  {
    return sum[find_set(i) - N];
  }

  int find_set(int i)
  {
    return (id[i] == i) ? i : (id[i] = find_set(id[i]));
  }

  bool connected(int i, int j)
  {
    return find_set(i) == find_set(j);
  }

  void union_set(int p, int q)
  {
    int i = find_set(p);
    int j = find_set(q);
    if (i == j) return;
    // make smaller root point to larger one
    if (sz[i - N] < sz[j - N]) { id[i] = j; sz[j - N] += sz[i - N]; sum[j - N] += sum[i
 - N]; }
    else                       { id[j] = i; sz[i - N] += sz[j - N]; sum[i - N] += sum[j
 - N]; }
    count--;
  }

  void move(int p, int q)
  {
    int i = find_set(p);
    int j = find_set(q);
    if (i == j) return;
    id[p] = j;
    sum[j - N] += p; sz[j - N]++;
    sum[i - N] -= p; sz[i - N]--;
  }
};

using namespace std;

int main(int argc, char *argv[])
{
  int N, M;
  while (scanf("%d␣%d", &N, &M) != EOF)
```

```
    {
      union_find uf(N);
      while (M--)
      {
        int command, p, q;
        scanf("%d", &command);
        if (command == 1) { scanf("%d␣%d", &p, &q); uf.union_set(p, q); }
        else if (command == 2) { scanf("%d␣%d", &p, &q); uf.move(p, q); }
        else { scanf("%d", &p); printf("%d␣%d\n", uf.size_set(p), uf.sum_set(p)); }
      }
    }
  return 0;
}
```

13 UVA 11690: Money Matters

```
class union_find
{
private:
  std::vector<int> id; // id[i] = parent of i
  std::vector<int> sz; // sz[i] = number of objects in subtree rooted at i
  int count; // number of components
public:
  union_find(int N)
    {
      count = N;
      id.assign(N, 1);
      sz.assign(N, 0);
      for (int i = 0; i < N; i++)
      {
        id[i] = i;
        sz[i] = 1;
      }
    }

    int nb_components()
    {
      return count;
    }

    int size_set(int i)
    {
      return sz[find_set(i)];
    }

    int find_set(int i)
    {
      return (id[i] == i) ? i : (id[i] = find_set(id[i]));
    }

    bool connected(int i, int j)
    {
      return find_set(i) == find_set(j);
    }

```

```cpp
    void union_set(int p, int q)
    {
      int i = find_set(p);
      int j = find_set(q);
      if (i == j) return;
      // make smaller root point to larger one
      if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
      else               { id[j] = i; sz[i] += sz[j]; }
      count--;
    }
};

using namespace std;

const int MAX_N = 10010;

int main(int argc, char *argv[])
{
  int TC, N, M;
  int debts[MAX_N];
  int balance[MAX_N];
  scanf("%d", &TC);
  while (TC--)
    {
      scanf("%d␣%d", &N, &M);
      union_find uf(N);
      for (int i = 0; i < N; ++i)
      {
        scanf("%d", &debts[i]);
      }
      while (M--)
      {
        int A, B;
        scanf("%d␣%d", &A, &B);
        uf.union_set(A, B);
      }
      memset(balance, 0, sizeof balance);
      for (int i = 0; i < N; ++i)
      {
        balance[uf.find_set(i)] += debts[i];
      }

      if (all_of(balance, balance + N, [](int v){ return v == 0; })) printf("POSSIBLE\n
");
      else printf("IMPOSSIBLE\n");
    }
  return 0;
}
```

14 UVA 12086: Potentiometers

```cpp
class segment_tree
{
private:
  std::vector<int> st, A;
```

```
int N;
inline int left (int p) { return p << 1; }
inline int right(int p) { return (p << 1) + 1; }

void build(int p, int L, int R)
{
  if (L == R)
    st[p] = A[L];
  else
    {
      build(left(p) , L              , (L + R) / 2);
      build(right(p), (L + R) / 2 + 1, R          );
      st[p] = st[left(p)] + st[right(p)];
    }
}

int find(int p, int L, int R, int i, int j)
{
  if (i >  R || j <  L) return 0;
  if (L >= i && R <= j) return st[p];

  int sum1 = find(left(p) , L              , (L+R) / 2, i, j);
  int sum2 = find(right(p), (L+R) / 2 + 1, R          , i, j);

  return sum1 + sum2;
}

int update_point(int p, int L, int R, int idx, int new_value)
{
  if (idx > R || idx < L)
    return st[p];

  if (L == idx && R == idx)
    {
      A[idx] = new_value;
      return st[p] = new_value;
    }

  int sum1 = update_point(left(p) , L              , (L + R) / 2, idx, new_value);
  int sum2 = update_point(right(p), (L + R) / 2 + 1, R          , idx, new_value);

  return st[p] = sum1 + sum2;
}

int update_range(int p, int L, int R, int i, int j, int new_value)
{
  if (i > R || j < L) return 0;

  if (L == R)
    {
      A[L] = new_value;
      return st[p] = new_value;
    }

  int sum1 = update_range(left(p) , L              , (L + R) / 2, i, j, new_value);
  int sum2 = update_range(right(p), (L + R) / 2 + 1, R          , i, j, new_value);

```

```
    return st[p] = sum1 + sum2;
  }

public:
  segment_tree(const std::vector<int>& _A)
  {
    A = _A; N = (int)A.size();
    st.assign(4 * N, 0);
    build(1, 1, N - 1);
  }

  int find(int i, int j)
  {
    return find(1, 1, N - 1, i, j);
  }

  int update_point(int idx, int new_value)
  {
    return update_point(1, 1, N - 1, idx, new_value);
  }

  int update_range(int i, int j, int new_value)
  {
    return update_range(1, 1, N - 1, i, j, new_value);
  }
};

int main(int argc, char *argv[])
{
  int N;
  int t = 0;
  while (scanf("%d", &N), t++, N) {
    if (t > 1) printf("\n");
    vector<int> resistances(N+1, 0);
    for (int i = 1; i <= N; ++i)
      {
        scanf("%d", &resistances[i]);
      }
    segment_tree st(resistances);
    char cmd[4];
    printf("Case␣%d:\n", t);
    while (scanf("␣%s", cmd), strcmp(cmd, "END")) {
      int a, b;
      scanf("%d␣%d", &a, &b);
      if (cmd[0] == 'S') {
        st.update_point(a, b);
      } else {
        printf("%d\n", st.find(a, b));
      }
    }
  }
  return 0;
}
```

15 UVA 10909: Lucky Number

```
const int N = 2000010;
bitset<N> lucky_numbers;

class segment_tree
{
private:
  std::vector<int> st;
  inline int left (int p) { return p << 1; }
  inline int right(int p) { return (p << 1) + 1; }

  void build(int p, int L, int R)
  {
    if (L == R)
      st[p] = lucky_numbers[L];
    else
      {
        build(left(p) , L, (L + R) / 2);
        build(right(p), (L + R) / 2 + 1, R);
        int n1 = st[left(p)], n2 = st[right(p)];
        st[p] = n1 + n2;
      }
  }

  int find(int p, int L, int R, int k)
  {
    if (L >= N) return -1;
    if (L == R) return L;
    int n1 = st[left(p)];
    if (n1 >= k) return find(left(p) , L              , (L+R) / 2, k);
    return find(right(p), (L+R) / 2 + 1, R          , k - n1);
  }

  void clear_point(int p, int L, int R, int k)
  {
    if (L == R)
      {
        lucky_numbers.reset(L);
        st[p] = 0;
        return;
      }
    int n1 = st[left(p)];
    if (n1 >= k)
      {
        clear_point(left(p), L, (L + R) / 2, k);
        n1 = st[left(p)];
      }
    else
      clear_point(right(p), (L + R) / 2 + 1, R          , k - n1);
    int n2 = st[right(p)];
    st[p] = n1 + n2;
  }

public:
  segment_tree()
  {
    st.assign(4 * N, 0);
  }

  void init()
  {
    build(1, 0, N - 1);
  }

  int find(int k)
  {
    return find(1, 0, N, k);
  }

  void clear_point(int k)
  {
    clear_point(1, 0, N - 1, k);
  }
};

void init(segment_tree& st)
{
  lucky_numbers.set();
  for (int i = 0; i < N; i += 2)
    {
      lucky_numbers.reset(i);
    }
  st.init();
  int n = N / 2 + 1;
  int k = 2;
  while (true)
    {
      int l = st.find(k++);
      if (l > n) break;
      int j = 0;
      for (int i = l; i <= n; i += l)
      {
        st.clear_point(i - j);
        ++j;
      }
      n -= j;
    }
}

int main(int argc, char *argv[])
{
  segment_tree st;
  init(st);
  int N;
  while (scanf("%d", &N) != EOF)
    {
      int L1 = -1, L2 = -1;
      if ((N & 1) == 0)
      {
        int i = N/2;
        for (; i > 0 && !lucky_numbers[i]; --i);
        for (; i > 0; i -= 2)
    {
     if (lucky_numbers[i] && lucky_numbers[N - i])
     {
      L1 = i;
      L2 = N - i;
      break;
     }
    }
   }
   if (L1 == -1) printf("%d␣is␣not␣the␣sum␣of␣two␣luckies!\n", N);
   else printf("%d␣is␣the␣sum␣of␣%d␣and␣%d.\n", N, L1, L2);
  }
 return 0;
}
```

16 UVA 750: 8 Queens Chess Problem

```
#include <cstdio>
#include <vector>
#include <cmath>

template <typename T>
struct domain
{
 std::vector<T> candidate;

 domain(int size) : candidate(size + 1)
 {
 }
 virtual bool is_a_solution(int k) = 0;
 virtual void process_solution(int k) = 0;
 virtual void next(std::vector<T>& possibilities, int k) = 0;
 virtual void make_move(int k)
 {
 }
 virtual void unmake_move(int k)
 {
 }
 virtual bool stop()
 {
  return false;
 }
 void set(int i, const T& elem)
 {
  candidate[i] = elem;
 }
};

template <typename T>
class backtracking
{
 domain<T>& dom;
 public:
 backtracking(domain<T>& dom) : dom(dom)
 {
}

 void backtrack(int k)
 {
  if (dom.is_a_solution(k))
  {
    dom.process_solution(k);
  }
  else
  {
    k++;
    std::vector<T> possibilities;
    dom.next(possibilities, k);
    for (int i = 0; i < possibilities.size(); ++i)
    {
      dom.set(k, possibilities[i]);
      dom.make_move(k);
      backtrack(k);
      dom.unmake_move(k);
      if (dom.stop()) return;
    }
  }
 }

 void execute()
 {
  backtrack(0);
 }
};

template<typename T>
backtracking<T> make_backtracking(domain<T>& dom)
{
 return backtracking<T>(dom);
}

using namespace std;

int a, b;

class height_queens : public domain<int>
{
 private:
 int size;
 int solution_count = 1;
 public:
 height_queens(int size) : domain(size)
 {
  this->size = size;
 }
 bool is_a_solution(int k)
 {
  return (k == size && candidate[b] == a);
 }
 void process_solution(int k)
 {
  printf("%2d      %d", solution_count++, candidate[1]);
  for (int i = 2; i <= size; i++)
  {
    printf("␣%d", candidate[i]);
  }
  printf("\n");
 }
 void next(std::vector<int>& possibilities, int k)
 {
  for (int i = 1; i <= size; i++)
  {
    bool legal_move = true;
    for (int j = 1; j < k; j++)
    {
      if (abs(k-j) == abs(i-candidate[j]) // diagonal threat
      || i == candidate[j]) // line threat
      {
        legal_move = false;
        break;
      }
    }
    if (legal_move)
    {
      possibilities.push_back(i);
    }
  }
 }
};

int main(int argc, char *argv[])
{
 int TC;
 scanf("%d", &TC);
 while (TC--)
 {
  scanf("%d␣%d", &a, &b);
  printf("SOLN␣␣␣␣␣␣␣COLUMN\n");
  printf("␣#␣␣␣␣␣␣1␣2␣3␣4␣5␣6␣7␣8\n\n");
  height_queens queens(8);
  auto bt = make_backtracking(queens);
  bt.execute();
  if (TC) printf("\n");
 }
 return 0;
}
```

17 UVA 357: Let Me Count the Ways

```
#include <cstdio>
#include <cstring>

using namespace std;

#define NB_COINS 5
#define MAX_MONEY 30000

```

```
long long ways[NB_COINS + 1][MAX_MONEY + 1];
int coins[NB_COINS + 1] = {0, 1, 5, 10, 25, 50};
int N;

long long solve(int index, int sum)
{
  if (sum < 0) return 0;
  if (index == NB_COINS + 1) return sum == 0;
  if (ways[index][sum] != -1) return ways[index][sum];
  long long res = solve(index, sum - coins[index]) + solve(index + 1, sum);
  return ways[index][sum] = res;
}

int main(int argc, char *argv[])
{
  memset(ways, -1, sizeof(ways));
  while (scanf("%d", &N) != EOF)
    {
      solve(1, N);
      if (ways[1][N] == 1) printf("There␣is␣only␣1␣way␣to␣produce␣%d␣cents␣change.\n",
N);
      else printf("There␣are␣%lld␣ways␣to␣produce␣%d␣cents␣change.\n", ways[1][N], N);
    }
  return 0;
}
```

```
#include <cstdio>
#include <cstring>

using namespace std;

#define NB_COINS 5
#define MAX_MONEY 30000

int main(int argc, char *argv[])
{
  int N;
  long ways[NB_COINS + 1][MAX_MONEY + 1];
  int coins[NB_COINS + 1] = {0, 1, 5, 10, 25, 50};
  memset(ways, 0, sizeof ways);
  ways[0][0] = 1;
  for (int i = 1; i <= NB_COINS; ++i)
  {
    ways[i][0] = 1;
    for (int j = 1; j <= MAX_MONEY; ++j)
    {
      long n = 0;
      if (j - coins[i] >= 0) n = ways[i][j - coins[i]];
      n += ways[i - 1][j];
      ways[i][j] = n;
    }
  }
  while (scanf("%d", &N) != EOF)
  {
    if (ways[5][N] == 1) printf("There␣is␣only␣1␣way␣to␣produce␣%d␣cents␣change.\n", N)
;
```

```cpp
    else printf("There␣are␣%ld␣ways␣to␣produce␣%d␣cents␣change.\n", ways[5][N], N);
  }
  return 0;
}
```

18 UVA 10131: Is Bigger Smarter?

```cpp
int main(int argc, char *argv[])
{
  vector<tuple<int, int, int>> elephants; //weight, IQ, index
  int lis[1001];
  int prev[1001];
  memset(prev, -1, sizeof prev);
  int W, IQ;
  int cpt = 0;
  while (scanf("%d␣%d", &W, &IQ) != EOF)
    {
      elephants.push_back(make_tuple(W, IQ, ++cpt));
    }
  sort(elephants.begin(), elephants.end());

  lis[0] = 1;
  int ans = 1, index = 0;
  for (int i = 1; i < elephants.size(); ++i)
    {
      int best_index = -1;
      lis[i] = 1;
      for (int j = 0; j < i; ++j)
      {
        if (get<0>(elephants[j]) != get<0>(elephants[i]) && get<1>(elephants[j]) > get<1>(elephants[i]) && 1 + lis[j] > lis[i])
        {
          lis[i] = 1 + lis[j];
          best_index = j;
        }
      }
      prev[i] = best_index;
      if (lis[i] > ans)
      {
        ans = lis[i];
        index = i;
      }
    }

  stack<int> s;
  s.push(get<2>(elephants[index]));
  while (prev[index] != -1)
    {
      index = prev[index];
      s.push(get<2>(elephants[index]));
    }
  printf("%d\n", ans);
  while (!s.empty())
    {
      printf("%d\n", s.top());
      s.pop();
    }
  return 0;
}
```

19 UVA 10536: Game of Euler

```
unsigned short board = 0;
//[15][14][13][12]
//[11][10][ 9][ 8]
//[ 7][ 6][ 5][ 4]
//[ 3][ 2][ 1][ 0]

unsigned short moves[] =
  {
    0x8000,
    0x4000,
    0x2000,
    0x1000,
    0x0800,
    0x0400,
    0x0200,
    0x0100,
    0x0080,
    0x0040,
    0x0020,
    0x0010,
    0x0008,
    0x0004,
    0x0002,
    0x0001,

    0xC000,
    0x3000,
    0x0C00,
    0x0300,
    0x00C0,
    0x0030,
    0x000C,
    0x0003,

    0x8800,
    0x4400,
    0x2200,
    0x1100,
    0x0088,
    0x0044,
    0x0022,
    0x0011,

    0xE000,
    0x7000,
    0x0E00,
    0x0700,
    0x00E0,
    0x0070,
    0x000E,
    0x0007,

    0x8880,
    0x4440,
    0x2220,
    0x1110,
    0x0888,
    0x0444,
    0x0222,
    0x0111
  };

const int NB_MOVES = 48;
char memo[1<<16][2];

bool is_terminal(unsigned short board)
{
  return board == 0xFFFF;
}

bool move_possible(int move, unsigned short board)
{
  return !(board & moves[move]);
}

int min_value(unsigned short board);
int max_value(unsigned short board)
{
  if (is_terminal(board)) return 1;
  if (memo[board][0] != 0) return memo[board][0];
  int v = -2;
  for (int i = 0; i < NB_MOVES; ++i)
    {
      if (move_possible(i, board))
      {
        v = max(v, min_value(board | moves[i]));
      }
    }
  return memo[board][0] = v;
}

int min_value(unsigned short board)
{
  if (is_terminal(board)) return -1;
  if (memo[board][1] != 0) return memo[board][1];
    int v = 2;
    for (int i = 0; i < NB_MOVES; ++i)
      {
        if (move_possible(i, board))
        {
          v = min(v, max_value(board | moves[i]));
        }
      }
    return memo[board][1] = v;
}
```

```

int main(int argc, char *argv[])
{
  int TC;
  scanf("%d", &TC);
  memset(memo, 0, sizeof memo);
  while (TC--)
    {
      board = 0;
      for (int i = 15; i >= 0; --i)
      {
        char c;
        scanf("␣%c", &c);
        board |= (c == 'X') << i;
      }
      if (max_value(board) == 1) printf("WINNING\n");
      else printf("LOSING\n");
    }
  return 0;
}
```

20 UVA 1213: Sum of Different Primes

```
const int MAX_N = 1121;
const int MAX_K = 15;

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

const int NB_PRIMES = sizeof(primes) / sizeof(int);

int memo[NB_PRIMES][MAX_K][MAX_N];

int nb_sums(int index, int nb_remaining, int sum)
```

```
{
  if (nb_remaining == 0) return sum == 0;
  if (index == NB_PRIMES) return 0;
  if (memo[index][nb_remaining][sum] != -1) return memo[index][nb_remaining][sum];
  return memo[index][nb_remaining][sum] =
    (sum - primes[index] < 0 ? 0 : nb_sums(index + 1, nb_remaining - 1, (sum - primes[index])))
    + nb_sums(index + 1, nb_remaining, sum);
}

int main(int argc, char *argv[])
{
  memset(memo, -1, sizeof memo);
  while (scanf("%d␣%d", &N, &K), (N || K))
    {
      printf("%d\n", nb_sums(0, K, N));
    }
  return 0;
}
```

21 UVA 11517: Exact Change

```
const int MAX_MONEY = 10010;

int main(int argc, char *argv[])
{
  int change[MAX_MONEY];
  int TC;
  scanf("%d", &TC);
  while (TC--)
    {
      int V, N;
      scanf("%d␣%d", &V, &N);
      fill(change, change + MAX_MONEY, INT_MAX);
      change[0] = 0;
      for (int i = 0; i < N; ++i)
      {
        int coin;
        scanf("%d", &coin);
        for (int j = MAX_MONEY - 1; j >= coin; --j)
        {
          if (change[j - coin] != INT_MAX) change[j] = min(change[j], 1 + change[j - coin]);
        }
      }
      for (int i = V; i < MAX_MONEY; ++i)
      {
        if (change[i] != INT_MAX)
        {
          printf("%d␣%d\n", i, change[i]);
          break;
        }
      }
    }
  return 0;
}
```

22 Hackerrank Dynamic Programming: Bricks Game

```
#include <algorithm>
#include <vector>
#include <iostream>
using namespace std;

int main(int argc, char *argv[])
{
  int TC;
  cin >> TC;
  while (TC--)
    {
      int N;
      cin >> N;
      vector<long long> bricks(N + 1, 0);
      for (int i = N; i >= 1; --i)
      {
        cin >> bricks[i];
      }
      vector<long long> sum(N + 1, 0);
      for (int i = 1; i <= N; ++i)
      {
        sum[i] += sum[i - 1] + bricks[i];
      }
      vector<long long> dp(N + 1, 0);
      dp[1] = sum[1];
      dp[2] = sum[2];
      dp[3] = sum[3];
      for (int i = 4; i <= N; ++i)
      {
        dp[i] = max(bricks[i] + sum[i - 1] - dp[i - 1],
                    max(bricks[i] + bricks[i - 1] + sum[i - 2] - dp[i - 2],
                        bricks[i] + bricks[i - 1] + bricks[i - 2] + sum[i - 3] - dp[i -
 3]));
      }
      cout << dp[N] << endl;
    }
  return 0;
}
```

Index