Competitive Programming Notebook

IllegalSkillsException

2018

Contents

1 Introduction

This document was written to be used in programming competitions by my team: *HammerHappy*. Conciseness (not clarity) was the priority.

2 Strings

2.1 Knuth-Morris-Pratt (KMP)

```
std::vector<int> compute_prefix(
                            const string& p) {
  int m = p.size();
  std::vector<int> pi(m);
  pi[0] = 0;

  int k = 0;
  for (int q = 1; q < m; q++) {
    while (k > 0 && p[k] != p[q])
      k = pi[k-1];
    if (p[k] == p[q])
      k++;
    pi[q] = k;
  }
  return pi;
}

void kmp_match(const string& s,
               const string& p) {
  std::vector<int> pi = compute_prefix(p);
  int q = 0;
  int n = s.size();
  int m = p.size();

  for (int i = 0;i < n;i++) {
    while (q > 0 && p[q] != s[i])
      q = pi[q - 1];
    if (p[q] == s[i])
      q++;
    if (q == m) {
      std::cout
              << "Match at pos: "
              << (i - m + 1)
              << std::endl;
    }
  }
}
```

2.2 Range Minimum Query

```
int main() {
  int N, Q, i, j, k;

  scanf("%d %d", &N, &Q);

  for (i = 0;i < N; i++)
    scanf("%d", &n[i]);

  for (i = 0; i < N; i++)
   m[i][0] = M[i][0] = n[i];

  for (i=1; (1 << i) <= N; i++) {
    for (j = 0; j + (1 << i) - 1 < N; j++) {
            m[j][i] = min(m[j][i - 1], m[j +
                      (1 << (i - 1))][i - 1]);
            M[j][i] = max(M[j][i - 1], M[j +
                       (1 << (i - 1))][i - 1]);
        }
  }

  for (k = 0;k < Q; k++) {
    scanf("%d %d",&i,&j);
    i--;
    j--;
    int t, p;
    t = (int)(log(j - i + 1) / log(2));
    p = 1 << t;
    printf("%d\n", max(M[i][t],
            M[j - p + 1][t])
            - min(m[i][t], m[j - p + 1][t]));
  }
  return 0;
}
```

$$M[i][j] = \max(M[i][j-1], M[i+2^{j-1}][j-1])$$

$$RMQ_A(i,j) = \max(M[i][k], M[j-2^k+1][k])$$

2.3 Nth Permutation

```
/**
 * Computes kth (0 to s.size()! - 1) permutation
 * of string s
```

```
  */
std::string nth_permutation(uint64_t k,
                            const std::string &s) {
   uint64_t factorial = 1;
   for (uint64_t i = 1; i <= s.size(); ++i) {
      factorial *= i;
   }

   std::string s_copy = s;
   std::string res;
   for (uint64_t j = 0; j < s.size(); ++j) {
      // compute how many permutations
      // on the rest of the
      // string s[j + 1 .. s.size() - 1]
      factorial /= s.size() - j;

      // store character
      uint64_t l = k / factorial;
      res += s_copy[l];

      // remove already used character
      s_copy.erase(s_copy.begin() + l);

      // compute new value of k
      k = k % factorial;
   }
   return res;
}
```

3 Dynamic Programming

3.1 Longest Common Subsequence (LCS)

```
int L[MAX][MAX] = {{0}};
int LCS(char A[], char B[]) {
  // m = strlen(A)
  // n = strlen(B)
  for (int i = m; i >= 0; i--) {}
    for (int j = n; j >= 0; j--) {
      if (!A[i] || !B[j])
        L[i][j] = 0;
      else if (A[i] == B[j])
        L[i][j] = 1 + L[i + 1][j + 1];
      else L[i][j] = max(L[i + 1][j],
                     L[i][j + 1]);
    }
  }
  return L[0][0];
}
```

```
int LCSString(int L[MAX][MAX]) {
  int i, j;
  i = j = 0;
  while (i < m && j < n) {
    if (A[i] == B[j]) {
      // put A[i] at the end
      // of solution string
      i++; j++;
    }
    if (L[i + 1][j] >= L[i][j + 1]) i++;
    else j++;
  }
}
```

3.2 Longest Increasing Subsequence (LIS)

3.2.1 $O(n^2)$ version

```
int pred[MAX_SIZE], lasti;
int LIS(int C[], int n) {
  int s[MAX_SIZE], max = INT_MIN;
  for (int i = 1; i < n; i++) {
    for (int j = 0; j < i; i++) {
      if (C[i] > C[j] && s[i] <= s[j]) {
        pred[i] = j;
        if ((s[i] = s[j] + 1) > max)
          lasti = i;
          max = s[i];
      }
    }
  }
  return max;
}

void PrintLIS() {
  int i, j, aux[MAX_SIZE];
  for (j = max - 1, i = lasti; j >= 0; j--) {
    aux[j] = C[i];
    i = pred[i];
  }

  for (j = 0;j < max; j++)
    printf(``%d\n'', aux[j]);
}
```

3.2.2 $O(n \log n)$ version

```
int a, num[120000], n, ans[120000], sz;

while (scanf("%d",&n) == 1){
  for (a = 0;a < n; a++)
```

```
    scanf("%d", &num[a]);
  sz = 0;
  for (a = 0;a < n; a++) {
    int* it = lower_bound(
                ans, ans + sz, num[a]);
    if (it != ans + sz) *it = num[a];
    else ans[sz++] = num[a];
  }
  printf("%d\n",sz);
}
```

3.3 MCM (Matrix Chain Multiplication)

```
void mcm() {
  int i, j, n = 3;
  for (i = 0;i < n; i++)
    m[i][i] = 0;

  for (i = n - 1; i >= 0; i--)
    for (j = i + 1; j <= n; j++)
      m[i][j] = calc(i, j);
}

int calc(int i, int j) {
  int res = INT_MAX;
  for (k = i; k < j; k++) {
    tmp = m[i][k] + m[k + 1][j]+
          Line[i] * Col[k] * Col[j];
    if (tmp < res) {
      res = tmp;
      s[i][j] = k;
    }
  }
  return res;
}

//printMCM(0,N-1);
void printMCM(int i, int j) {
  if (i == j) printf("A%d",i);
  else {
    putchar('(');
    printMCM(i, s[i][j]);
    putchar('*');
    printMCM(s[i][j] + 1, j);
    putchar(')');
   }
}
```

3.4 Knapsack

```
int n[WSIZE][ISIZE] = {{0}}
// put one zero in weight and value;
// e.g.
// weight={>0<,3,4,5}
// value={>0<,3,4,5,6};
int knapsack(int items, int W,
          int value[], int weight[]){
  for (int i = 1;i <= items; i++) {
    for (int j = 0; j <= W; j++) {
      if (weight[i] <= j) {
        if (value[i] + n[i-1][j-weight[i]]
          > n[i-1][j]) {
          n[i][j] = value[i] +
                    n[i-1][j-weight[i]];
        } else {
          n[i][j]=n[i-1][j];
        }
      } else n[i][j]=n[i-1][j];
    }
  }
  return n[items][W];
}

void print_sequence(int items, int W, int weight[]) {
  int i = items, k = W;
  while (i > 0 && k > 0) {
    if (n[i][k] != n[i-1][k]) {
      printf("item_%d_is_in\n", i);
      k = k-weight[i-1];
    }
    i--;
  }
}
```

3.5 Counting Change

```
int coins[] = {50,25,10,5,1};
int coin_change(int n) {
  table[0] = 1;
  for (int i = 0; i < 5; i++) {
    c = coins[i];
    for (int j = c; j <= n; j++)
      table[j] += table[j - c];
  }
  return table[n];
}
```

3.6 Coin Changing

```
int n[10000], i, N;
int coins[]={50,25,10,5,1},k;

scanf("%d", &N);
for (int i = 0; i <= N; i++)
```

```
  n[i] = INT_MAX;
n[0] = 0;
for (int i = 0; i < 5; i++) {
  for (k = 0;k <= N - coins[i]; k++) {
    n[k + coins[i]] =
      min(n[k] + 1, n[k + coins[i]]);
  }
}
printf("%d\n", n[N]);
```

3.7 Biggest Sum

```
#define SIZE 20000
int n[SIZE];

int biggest_sum() {
  int k, s, b;
  int xl, xr, best, prevx;

  cin>>k;
  for (int i = 1; i <= k; i++) {
    xr = xl = 0;

    cin >> s;
    for (int j = 0; j < s - 1; j++)
      cin >> n[j];

    prevx = xl = xr = 0;
    best = b = n[0];
    for (int j = 1; j < s - 1; j++) {
      if (b < 0)
        prevx = j;
      b = n[j] + max(0, b);
      if (b > best ||
        (b == best &&
            j - prevx > xr - xl)) {
        xl = prevx;
        xr = j;
        best = b;
      }
    }
    if (best > 0)
      cout << "Biggest sum " << i
           << is between "<< xl + 1
           << and " << xr + 2
           << endl;
  }
  return 0;
}
```

3.8 Edit Distance

Possible actions:

1. Delete a character
2. Insert a new character
3. Replace a character

```
int edit_distance(char *str1, char *str2) {
  int n[SIZE][SIZE];
  int i, j, value;

  for (i = 0; i <= str1_len; i++) n[i][0] = i;
  for (j = 0; j <= str2_len; j++) n[0][j] = j;

  for (i = 1; i <= str1_len; i++) {
    for (j = 1; j <= str2_len; j++) {
      value = (str1[i - 1] != str2[j - 1]);

      n[i][j] = min(n[i - 1][j - 1] + value,
                n[i - 1][j] + 1,
                n[i][j - 1] + 1);
    }
  }
  return n[str1_len][str2_len];
}
T(i,j) = min(C_d + T(i-1,j),
         T(i, j-1) + C_i,
         T(i-1, j-1) + (A[i]==B[j] ? 0 : C_r))
```

3.9 Integer Partitions

$P(n)$ represents the number of possible partitions of a natural number n. $P(4) = 5, 4, 3+1, 2+2, 2+1+1, 1+1+1+1$
$P(0) = 1$
$P(n) = 0, n < 0$
$P(n) = p(1, n)$
$p(k, n) = p(k+1, n) + p(k, n-k)$
$p(k, n) = 0$ if $k > n$
$p(k, n) = 1$ if $k = n$

3.10 Box Stacking

A set of boxes is given. $Box_i = h_i, w_i, d_i$.
We can only stack box i on box j if $w_i < w_j$ and $d_i < d_j$.
To consider all the orientations of the boxes, replace each box with 3 boxes such that $w_i \leq d_i$ and $box_1[0] = h_i, box_2[0] = w_i, box_3[0] = d_i$.
Then, sort the boxes by decreasing area($w_i * d_i$).
$H(j)$ =tallest stack of boxes with box j on top.

$H(j) = \max_{i<j \& w_i>w_j \& d_i>dj}(H(i)) + h_j$
Check $H(j)$ for all values of j.

3.11 Building Bridges

Maximize number of non-crossing bridges. Ex:
bridge1:$2, 5, 1, n, \cdots, 4, 3$
bridge2:$1, 2, 3, 4, \cdots, n$
Let $X(i)$ be the index of the corresponding city on northern bank. $X(1) = 3, X(2) = 1, \ldots$.
Find longest increasing subsequence of $X(1), \cdots, X(n)$.

3.12 Partition Problem

Input: A given arrangement S of non-negative numbers $s_1, \ldots, s_n$ and an integer k.
Output: Partition S into k ranges, so as to minimize the maximum sum over all the ranges.

```
int M[1000][100], D[1000][100];
void partition_i(vector<int> &v, int k) {
  int p[1000], n = v.size();
  v.insert(v.begin(),0);
  p[0] = 0;
  for(int i = 1;i < v.size(); i++)
    p[i] = p[i - 1] + v[i];

  for (int i = 1; i <= n; i++)
    M[i][1]=p[i];
  for (int i = 1; i <= k; i++)
    M[1][i] = v[1];
  for (int i = 2; i <= n; i++) {
    for (int int j = 2; j <= k; j++) {
      M[i][j] = INT_MAX << 1 - 1;
      int s = 0;
      for (int x = 1; x <= i - 1; x++) {
        s = max(M[x][j - 1], p[i] - p[x]);
        if (M[i][j] > s) {
          M[i][j] = s;
          D[i][j] = x;
        }
      }
    }
  }
  printf("%d\n", M[n][k]);
}

//n = number of elements of the initial set
void reconstruct_partition(
    const vector<int> &S, int n, int k) {
  if (k == 1) {
    for (int i = 1; i <= n; i++)
      printf("%d ", &S[i]);
    putchar('\n');
  } else {
    reconstruct_partition(S, D[n][k], k - 1);
    for (int i = D[n][k] + 1; i<= n; i++)
      printf("%d ", S[i]);
    putchar('\n');
  }
}
```

3.13 Balanced Partition

```
enum {DONT_GET, GET};
char **sol, **P;

// return 1 if there is a subset
// of v0...vi with sum j
// 0 otherwise
int calcP(int i, int j, const vi &v) {
  if (i < 0 || j < 0) return 0;
  if (P[i][j] != -1) return P[i][j];

  if (j == 0) { // trivial case
    sol[i][j] = DONT_GET;
    return P[i][j] = 1;
  }
  if (v[i] == j) {
    sol[i][j] = GET;
    return P[i][j] = 1;
  }

  int res1 = calcP(i - 1, j, v);
  int res2 = calcP(i - 1, j - v[i], v);
  if (res1 >= res2)
    P[i][j] = res1, sol[i][j] = DONT_GET;
  else P[i][j] = res2, sol[i][j] = GET;
  return P[i][j];
}


// v is the vector of values
// k is the maximum value in v
// sum is the sum of all elements in v
void balanced_partition(vi &v,
                              int k, int sum) {
  P = new char*[v.size()];
  sol = new char*[v.size()];
  for (int i = 0; i < v.size(); i++) {
    P[i] = new char[k * v.size() + 1];
    sol[i] = new char[k * v.size() + 1];
    for (int j = 0;
          j < k * v.size() + 1; j++)
      P[i][j] = -1, sol[i][j] = DONT_GET;
  }
  for (int i = 0; i < v.size(); i++)
    for (int j = 0;
          j < v.size() * k + 1; j++)
      calcP(i, j, v);
  //calcP(v.size() - 1, sum/2, v);

  int S = sum / 2;
  if (sum & 1 || !P[v.size() - 1][S])
    cout << "ERROR" <<endl;
  else cout << "SUCCESS" << endl;
}

void free_mem(vi& v) {
   for (int i = 0; i < v.size(); i++) {
     delete P[i]; delete sol[i];
   }
   delete[] P;
   delete[] sol;
}

// get_solution(v.size() - 1,
// accumulate(v.begin(), v.end(), 0) / 2,
// v1, v2, v);
void get_solution(int i, int j,
            vi &S1, vi &S2, vi &v) {
  if (j < 0 || i < 0) return;
  if (sol[i][j] == GET) {
    S1.push_back(v[i]);
    return get_solution(i - 1, j - v[i],
                              S1 ,S2, v);
  } else {
    S2.push_back(v[i]);
    return get_solution(i - 1, j,
                              S1, S2, v);
  }
}
```

4 Graphs

4.1 Heap

```
#define LEFT(i) (2 * (i + 1) - 1)
#define RIGHT(i) (2 * (i + 1))
#define PARENT(i) (((i) + 1) / 2 - 1)

int *min_heap, *heap_place;
long int *keys;
int heap_size=0;

#define update_place(i) \
      heap_place[min_heap[(i)]]=(i)

void init_heap(int nelems) {
  min_heap = new int[nelems];
  keys = new long int[nelems]
  heap_place = new int[nelems];
  heap_size = nelems;
  for (int i = 0; i < nelems; i++) {
    min_heap[i] = i;
    heap_place[i] = i;
    keys[i] = LONG_MAX;
  }
}

void heap_min_heapify(int i) {
  int smallest, temp;
  int l = LEFT(i);
  int r = RIGHT(i);

  if (l < heap_size
     && keys[min_heap[l]]
       < keys[min_heap[i]])
    smallest = l;
  else smallest = i;

  if (r < heap_size &&
       keys[min_heap[r]]
       < keys[min_heap[smallest]])
    smallest = r;

  if (smallest!=i) {
    temp = min_heap[i];
    min_heap[i] = min_heap[smallest];
    min_heap[smallest] = temp;
    update_place(smallest);
    update_place(i);
    heap_min_heapify(smallest);
  }
}


int heap_extract_min() {
  if (heap_size < 1)
    return -1;
  int res = min_heap[0];
  heap_size--;
  min_heap[0] = min_heap[heap_size];
  update_place(0);
  heap_min_heapify(0);
  return res;
}


void heap_decrease_key(int elem,
                       long int key) {
  int i = heap_place[elem];

  keys[min_heap[i]]=key;

  while (i > 0
       && keys[min_heap[PARENT(i)]] >
    keys[min_heap[i]]) {
    int temp = min_heap[i];
    min_heap[i] = min_heap[PARENT(i)];
    min_heap[PARENT(i)] = temp;
    update_place(i);
    update_place(PARENT(i));
    i = PARENT(i);
  }
}
```

4.2 Find an Eulerian Path

```
stack<int> s;
vector<list<int>> adj;

void remove_edge(int u, int v) {
  for (list<int>::iterator it = adj[u].begin();
          it != adj[u].end(); it++) {
    if (*it == v) {
      it = adj[u].erase(it);
      return;
    }
  }
}

int path(int v) {
  int w;
  for (; adj[v].size(); v = w) {
    s.push(v);
    list<int>::iterator it = adj[v].begin();
    w = *it;;
    remove_edge(v,w);
    remove_edge(w,v);
    edges--;
  }
  return v;
}

//u - source, v-destiny
int eulerian_path(int u, int v) {
  printf("%d\n", v);
  while (path(u) == u && !s.empty()) {
    printf("-%d", u = s.top());
          s.pop();
  }
  return edges == 0;
}
```

4.3 Breadth First Search

```
bool adj[N][N];
int colour[N], d[N], p[N];
void bfs() {
  queue<int> q;
  int source = 0;

  for (int i = 0; i < N; i++) {
    d[i] = INF;
    p[i] = -1;
    colour[i] = WHITE;
  }

  d[source] = 0;
  colour[source] = GRAY;
  q.push(source);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int v = 0;v < N; v++) {
      if (colour[v] == WHITE
          && adj[u][v]) {
        colour[v] = GRAY;
        d[v] = d[u]+1;
        p[v] = u;
```

```
            q.push(v);
        }
    }
    colour[u] = BLACK;
  }
}
```

4.4 DFS/TopSort

$O(V+E)$

Recursive:

```
void dfs(int u) {
  colour[u] = GRAY;
  for (int v = 0;v < N; v++) {
    if (colour[v] == WHITE && adj[u][v]) {
      p[v] = u;
      dfs(v);
    }
  }
  colour[u] = BLACK;
  //put node in front of a list if topsort
}
```

Iterative:

```
typedef enum {WHITE, GRAY, BLACK} color_t;
vector<color_t> color;
// SCC: vector<int> close_time;

stack<int> dfs;
color = vector<color_t>(N, WHITE);
for (int i = 0; i < N; i++) {
  if (color[i] != WHITE)
    continue;
  dfs.push(i);
  // SCC2: for (int i = N - 1; i >= 0; i--) {
  // SCC2: if (color[close_time[i]] != WHITE)
  //         continue;
  // SCC2: dfs.push(close_time[i]);
  while (!dfs.empty()) {
    int u = dfs.top();
    switch(color[u]) {
      case WHITE:
        color[u] = GRAY;
        for (v in adj[u]) {
          // SCC2: for (v in adj_t[u]) {
          if (color[v] == WHITE) {
            dfs.push(v);
          }
        }
        break;
      case GRAY:
        color[u]=BLACK;
        dfs.pop();
        // put node in front of a list
        // if topsort
        // SCC1: close_time.push_back(u);
        break;
      case BLACK:
        dfs.pop();
        break;
    }
  }
}
```

Maximum Spanning Tree:
Negate all the edge weights and determine the minimum spanning tree.
Minimum Product Spanning Tree:
Replace all the edge weights with their logarithm
Strongly Connected Components:

1. Run DFS: Save closing times of all vertexes.
2. Compute adj_t.
3. Run DFS: Reverse order of closing times. In adj_t.
4. Each resulting tree is a SCC.

4.5 Prim's Algorithm

4.5.1 Naive version

```
double Prim(int start, int nvert) {
  bool in[N];
  double dist[N];
  int p[N], v;

  for (int i = 0; i < nvert; i++) {
    in[i] = false;
    dist[i] = INT_MAX;
    p[i] = -1;
  }

  dist[start] = 0;
  v = start;
  while (!in[v]) {
    in[v] = true;
    for (int i = 0; i < nvert; i++) {
      if (adj[v][i] && !in[i]) {
        if (dist[i] > adj[v][i]) {
          dist[i] = adj[v][i];
        p[i] = v;
      }
    }
  }

  double d = FLT_MAX;
  for (int i = 0;i < nvert; i++) {
    if (!in[i] && d > dist[i]) {
      v = i;
      d = dist[i];
    }
  }
}
double res = 0;
for (int i = 0;i < nvert; i++)
  res += dist[i];
return res;
}
```

4.5.2 Set version

```
std::vector<std::pair<int,int> > graph[SIZE];

struct cmp_fn {
  bool operator() (const int&a,
                   const int &b) const {
    return dist[a] < dist[b] ||
        (dist[a] == dist[b] && a < b);
  }
};
int Prim(int start, int nvert) {
  set<int, cmp_fn> s;
  dist[start] = 0;
  s.insert(start);

  while (s.size()) {
    int v = *(s.begin());
    s.erase(s.begin());

    in[v] = true;
    for (unsigned int i = 0;
         i < graph[v].size(); i++) {
      int node = graph[v][i].first;
      int length = graph[v][i].second;
      if (!in[node]) {
        if (dist[node] > length) {
          if (s.find(node) != s.end())
              s.erase(s.find(node));
          dist[node] = length;
        }
          s.insert(node);
        }
      }
    }

  ull res = 0;
  for (int i = 0;i < nvert; i++)
    res += dist[i];
  return res;
}
```

4.6 Dijkstra

4.6.1 Naive version

```
int dijkstra(int source, int dest,
             int nvert, int d[], int p[]) {
  bool in[N];
  int u;

  for (int i = 0; i < nvert; i++) {
    in[i] = false;
    d[i] = INF;
    p[i] = -1;
  }
  d[source] = 0;
  u = source;
  while (!in[u]) {
    in[u] = true;
    for (int v = 0; v < nvert; v++) {
      if (adj[u][v]
          && d[v] > d[u] + adj[u][v]) {
        p[v] = u;
        d[v] = d[u] + adj[u][v];
      }
    }

    int dist = INT_MAX;
    for (int i = 0; i < nvert; i++) {
      if (!in[i] && d[i] < dist) {
        u = i;
        dist = d[i];
      }
    }
  }
  return d[dest];
}
```

4.6.2 Set version

```
#define VPI std::vector<std::pair<int,int>>
```

```
int dijkstra(const VPI graph[SIZE],
             int S, int T) {
  dist[S] = 0;

  set<Node> s;
  s.insert(Node(S));

  int u = S;
  while (s.size()) {
    Node n = *(s.begin());
    s.erase(s.begin());
    u = n.x;
    seen[u] = true;

    unsigned int i;
    for (i = 0; i < graph[u].size(); ++i) {
      int node = graph[u][i].first;
      int lat = graph[u][i].second;
      if (!seen[node]
        && dist[node] > dist[u] + lat) {
        if (s.find(Node(node))
                      != s.end()) {
          s.erase(s.find(Node(node)));
        }
        dist[node] = dist[u] + lat;
        s.insert(Node(node));
      }
    }
  }

  return dist[T];
}
```

4.7 Kth Shortest Paths $O(Km)$

```
/*
 * u - source node
 * p - predecessor vector
 * h - vector of transformation
 * v - result vector
 */
 #define vvi vector<vector<int> >
void path(int u, const vector<int> &p,
       const vector<int> &h,
    vector<int> &v) {
  if (u != -1) {
    path(p[u], p, h, v);
    v.push_back(h[u]);
  }
}

vvi dijkstra(int source, int dest, int K) {
  vector<int> count(SIZE), d(10000),
           p(10000), h(10000), X;
  vvi res;

  for (int i = 0; i < N; i++)
           p[i] = -1;
  int elm = 1;
  h[elm] = source;
  d[elm] = 0;
  X.push_back(elm);

  while (count[dest] < K && !X.empty()) {
    int ind = 0;
    for (unsigned int i = 1;
         i < X.size(); i++) {
    if (d[X[i]] < d[X[ind]])
      ind = i;
    }
    int k = X[ind];
    X.erase(X.begin() + ind);
    int i = h[k];

    count[i]++;
    if (i == dest) {
      vector<int> v;
      path(k, p, h, v);
      res.push_back(v);
    }

    if (count[i] <= K) {
      for (int j = 0;
           j < SIZE; j++) {
        if (adj[i][j]) {
          elm++;
          d[elm] = d[k] + adj[i][j];
          p[elm] = k;
          h[elm] = j;
          X.push_back(elm);
        }
      }
    }
  }
  return res;
}
```

4.8 Floyd-Warshall $O(n^3)$

```
void floyd(int adj[NVERT][NVERT]) {
  for (int k = 1;k <= NVERT; k++) {
    for (int i = 1;i <= NVERT; i++) {
      for (int j = 1;j <= NVERT; j++) {
        int through_k = adj[i][k]
                   + adj[k][j];
        if (through_k < adj[i][j])
          adj[i][j] = through_k;
      }
    }
  }
}
```

4.9 Bellman-Ford

```
typedef struct {
  int source;
  int dest;
  int weight;
} Edge;

void BellmanFord(Edge edges[], int edgecount,
            int nodecount, int source) {
  int *distance = new int[nodecount];
  for (int i=0; i < nodecount; i++)
    distance[i] = INT_MAX;

  // source node distance is set to zero
  distance[source] = 0;

  for (int i = 0; i < nodecount; i++) {
    for (int j = 0; j < edgecount; j++) {
      if (distance[edges[j].source]
                      != INT_MAX) {
        int new_distance =
             distance[edges[j].source] +
             edges[j].weight;

        if (new_distance <
             distance[edges[j].dest])
          distance[edges[j].dest] =
                      new_distance;
      }
    }
  }

  for (int i = 0; i < edgecount; i++) {
    if (distance[edges[i].dest] >
      distance[edges[i].source] +
            edges[i].weight) {
      puts("Negative edge weight
            cycles detected!");
      free(distance);
      return;
    }
  }

  for (int i = 0; i < nodecount; i++) {
    printf("The shortest distance between
            nodes %d and %d is %d\n",
          source, i, distance[i]);
  }
  delete[] distance;
}
```

4.10 Detecting Bridges

```
int dfs(int u, int p) {
  colour[u] = 1;
  dfsNum[u] = num++;
  int leastAncestor = num;
  for (int v = 0; v < N; v++) {
    if (M[u][v] && v!=p) {
      if (colour[v] == 0) {
        int rec = dfs(v,u);
        if (rec > dfsNum[u])
          cout << "Bridge: "
               << u <<" " << v
               << endl;
        leastAncestor =
          min(leastAncestor,rec);
      }
      else {
        leastAncestor = min(leastAncestor,
                      dfsNum[v]);
      }
    }
  }
  colour[u] = 2;
  return leastAncestor;
}
```

4.11 Finding a Loop in a Linked List $O(n)$

```
function boolean hasLoop(Node startNode) {
  Node slowNode, fastNode1, fastNode2;
  slowNode = fastNode1 = fastNode2 = startNode;
  while (slowNode && fastNode1 = fastNode2.next()
          && fastNode2 = fastNode1.next()) {
    if (slowNode == fastNode1 ||
      slowNode == fastNode2)
        return true;
    slowNode = slowNode.next();
  }
  return false;
}
```

4.12 Tree diameter

Pick a root and start a DFS from it which returns both the diameter of the subtree and its maximum height. The diameter is the maximum of (left diameter, right diameter, left height + right height).

4.13 Union Find

```
int Rank[SIZE];
int P[SIZE];

void create_set(int x) {
  P[x] = x;
  Rank[x] = 0;
}

void merge_sets(int x, int y) {
  int px = find_set(x);
  int py = find_set(y);
  if (Rank[px] > Rank[py])
    P[py] = px;
  else P[px] = py;

  if (Rank[px] == Rank[py])
    Rank[py]++;
}

int find_set(int x) {
  if (x != P[px])
    P[x] = find_set(P[x]);
  return P[x];
}

void connected_components() {
  for each vertex i
    do create_set(i);

  for each edge (u,v)
    if (find_set(u) != find_set(v))
      merge_sets(u,v);
}

bool same_conponents(int u,int v) {
  if (find_set(u) == find_set(v))
    return true;
  else return false;
}
```

4.14 Edmonds Karp

```
struct edge {
   int dest;
   int max_weight;
   int flow;
   edge * residual;
}

int nnodes;

typedef map<int, edge*> node;
typedef node** graph;
graph grafo;

void create_edge(int source,
            int dest, int weight) {
  if ((*grafo[source]).find(dest) ==
        (*grafo[source]).end()) {
    edge* e = new edge;
    edge* res = new edge;
    e->dest = dest;
    res->dest=source;
    e->max_weight=weight;
    res->max_weight=weight;
    e->flow=0;
    res->flow=weight;
    e->residual=res;
    res->residual=e;
    (*grafo[source])[dest] = e;
    (*grafo[dest])[source] = res;
    return;
  }

  edge* e = (*grafo[source])[dest];
  edge* res = e->residual;
  e->max_weight += weight;
  res->max_weight += weight;
  res->flow += weight;
}

int update_path(int flowsource,
                int flowdest) {
  int flow = INT_MAX;
  int noded = flowdest;
  while (noded != flowsource) {
    int source=p[noded];
    edge* e=(*grafo[source])[noded];
    if (flow>e->max_weight-e->flow) {
      flow=e->max_weight-e->flow;
    }
    noded=source;
  }

  noded=flowdest;
  while (noded != flowsource) {
    int source = p[noded];
    edge* e = (*grafo[source])[noded];
    e->flow+=flow;
    e->residual->flow-=flow;
    noded=source;
  }
  return flow;
}

int edmonds_karp(int source, int dest) {
  int res = 0;
  while (1) {
    bfs(source);
    if (colour[dest] == WHITE) {
      return res;
    }
    res += update_path(source,dest);
  }
  return res;
}
```

4.15 Ford Fulkerson

```
#define V 110
int graph[V][V];

bool bfs(int rGraph[V][V], int s,
         int t, int parent[]) {
  // Create a visited array and
  // mark all vertices as not visited
  bool visited[V];
  memset(visited, 0, sizeof(visited));

  // Create a queue
  // enqueue source vertex and
  // mark source vertex as visited
  std::queue <int> q;
  q.push(s);
  visited[s] = true;
  parent[s] = -1;

  // Standard BFS Loop
  while (!q.empty()) {
    int u = q.front();
    q.pop();

    for (int v = 0; v < V; v++) {
      if (visited[v] == false
         && rGraph[u][v] > 0) {
        q.push(v);
        parent[v] = u;
        visited[v] = true;
      }
    }
  }

  // If we reached sink in BFS starting from
  // source, then return true, else false
  return (visited[t] == true);
}
```

4.16 Widest path problem

In an undirected graph, a widest path may be found as the path between the two vertices in the maximum spanning tree of the graph

5 Geometrical Algorithms

5.1 Circle

Formula is given by

$$x^2 + y^2 = r^2$$

5.2 Triangle's medians

Any triangle's area T can be expressed in terms of its medians m_a, m_b, m_c as follows. Denoting their semi-sum $(ma + mb + mc)/2$ as s, we have

$$A = \frac{4}{3}\sqrt{s(s-m_a)(s-m_b)(s-m_c)}$$

The sides of the triangle are given, from the medians:

$$a = \frac{2}{3}\sqrt{-m_a^2 + 2m_b^2 + 2m_c^2}$$
$$b = \frac{2}{3}\sqrt{-m_b^2 + 2m_a^2 + 2m_c^2}$$
$$c = \frac{2}{3}\sqrt{-m_c^2 + 2m_b^2 + 2m_a^2}$$

5.3 Heron's formula

$$s = \frac{a+b+c}{2}$$

Area is given by

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

5.4 Dot Product

```
int dot(int[] A, int[] B, int[] C) {
  int AB[2], BC[2];
  AB[0] = B[0]-A[0];
  AB[1] = B[1]-A[1];
  BC[0] = C[0]-B[0];
  BC[1] = C[1]-B[1];
  int dot = AB[0] * BC[0] + AB[1] * BC[1];
  return dot;
}
```

5.5 Cross Product

```
int cross(int[] A, int[] B, int[] C) {
  int AB[2], AC[2];
  AB[0] = B[0]-A[0];
  AB[1] = B[1]-A[1];
  AC[0] = C[0]-A[0];
  AC[1] = C[1]-A[1];
  int cross = AB[0] * AC[1] - AB[1] * AC[0];
  return cross;
}
```

5.6 Point on segment

A point is on a segment if its distance to the segment is 0.

Given two different points (x_1, y_1) and (x_2, y_2) the values of A,B, and C for $Ax + By + C = 0$ are given by

$$A = y_2 - y_1$$
$$B = x_1 - x_2$$
$$C = A * x_1 + B * y_1$$

5.7 Intersection of segments

```
double det = A1*B2 - A2*B1
if (det == 0) {
  //Lines are parallel
} else {
  double x = -(A1*C2 - A2*C1) / det
  double y = -(B1*C2 - B2*C1) / det
}
```

5.8 Position of point in relation to line

```
//Input: three points P0, P1, and P2
//Return: >0 for P2 left of the line through P0 and P1
//    = 0 for P2 on the line
//    < 0 for P2 right of the line
int isLeft( Point P0, Point P1, Point P2 ) {
  return ( (P1.x - P0.x) * (P2.y - P0.y)
      - (P2.x - P0.x) * (P1.y - P0.y) );
}
```

5.9 Distance between point and line/segment

If the line is in the form $Ax + By + C = 0$:

$$d = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$$

```
//Compute the dist. from AB to C
//if isSegment=strue, AB is a seg., not a line.
double linePointDist(int[] A, int[] B,
      int[] C, boolean isSegment) {
  double dist = cross(A,B,C) / distance(A,B);
  if (isSegment) {
    int dot1 = dot(A,B,C);
    if (dot1 > 0)
      return distance(B,C);
    int dot2 = dot(B,A,C);
    if (dot2 > 0)
      return distance(A,C);
  }
  return abs(dist);
}
```

5.10 Polygon Area

```
int area = 0;
/*int N = lengthof(p);*/

for (int i = 1; i + 1 < N; i++) {
  int x1 = p[i][0] - p[0][0];
  int y1 = p[i][1] - p[0][1];
  int x2 = p[i+1][0] - p[0][0];
  int y2 = p[i+1][1] - p[0][1];
  int cross = x1*y2 - x2*y1;
  area += cross;
}
return fabs(area/2.0);
```

5.11 Convex Hull

```
#include <vector>
vector<point> ConvexHull(vector<point> P) {
  int n = P.size(), k = 0;
  vector<point> H(2*n);

  // Sort points lexicographically
  sort(P.begin(), P.end());

  // Build lower hull
  for (int i = 0; i < n; i++) {
    while (k >= 2
       && cross(H[k-2], H[k-1], P[i]) <= 0)
      k--;
    H[k++] = P[i];
  }

  // Build upper hull
  for (int i = n-2, t = k+1; i >= 0; i--) {
    while (k >= t
        && cross(H[k-2], H[k-1], P[i]) <= 0)
      k--;
    H[k++] = P[i];
  }

  H.resize(k);
  return H;
}
```

5.12 Closest pair of points

```
double delta_m(vp &ql,vp &qr, double delta) {
  uint64_t j = 0;
  double dm = delta;
  for (uint64_t i = 0; i < ql.size(); i++) {
    point p = ql[i];

    while (j < qr.size()
        && qr[j].y < p.y - delta)
      j++;

    uint64_t k = j;
    while (k < qr.size()
         && qr[k].y <= p.y + delta) {
      dm = min(dm, dist(p, qr[k]));
      k++;
    }
  }
  return dm;
}

vp select_candidates(vp &p, int l, int r,
      double delta, double midx) {
  vp n;
  for (int i = l;i <= r; i++) {
    if (abs(p[i].x - midx) <= delta)
      n.push_back(p[i]);
  }
  return n;
}

double closest_pair(vp &p, int l, int r) {
  if (r - l + 1 < 2) return INT_MAX;
  int mid = (l + r) / 2;
  double midx = p[mid].x;
  double dl = closest_pair(p, l, mid);
  double dr = closest_pair(p, mid + 1, r);
  double delta = min(dl, dr);

  vp ql, qr;
  ql = select_candidates(p, l,
                         mid, delta, midx);
  qr = select_candidates(p, mid + 1,
                         r, delta, midx);

  double dm = delta_m(ql, qr, delta);

  vp res;
  merge(p.begin() + l,p.begin() + mid + 1,
     p.begin() + mid + 1, p.begin() + r + 1,
       back_inserter(res), cmp);
  copy(res.begin(), res.end(), p.begin() + l);
  return min(dm, min(dr, dm));
}
```

5.13 Test if point is inside a polygon

```
int wn_PnPoly(Point P, Point* V, int n) {
  int wn = 0; // the winding number counter

  // loop through all edges of the polygon
  for (int i = 0; i < n; i++) {
    if (V[i].y <= P.y) {
      if (V[i + 1].y > P.y)
        if (isLeft(V[i],
             V[i + 1], P) > 0)
          ++wn;
    } else {
      if (V[i+1].y <= P.y)
        if (isLeft(V[i],
             V[i+1], P) < 0)
          --wn;
    }
  }
  return wn;
}
```

5.14 Circle from 3 points

```
int main() {
  double ax, ay, bx, by, cx, cy, xres, yres;
  double xmid,ymid,A1,B1,C1,A2,C2,B2,dist;

  while (scanf("%lf %lf %lf %lf %lf %lf",
        &ax,&ay,&bx,&by,&cx,&cy)==6) {
    A1 = by - ay;
    B1 = ax - bx;
    xmid = min(ax, bx) + (max(ax, bx)
           - min(ax, bx)) / 2.0;
    ymid = min(ay, by) + (max(ay, by)
           - min(ay, by)) / 2.0;
    C1 = -B1 * xmid + A1 * ymid;

    B2 = bx - cx;
    A2 = cy - by;
    xmid = min(bx, cx) + (max(bx, cx)
           - min(bx, cx)) / 2.0;
    ymid = min(by, cy) + (max(by, cy)
           - min(by, cy)) / 2.0;
    C2 = -B2 * xmid + A2 * ymid;

    //intersection of segments
    intersection(A1, B1, C1, A2,
                 B2, C2, &xres, &yres);
    dist = sqrt(pow(xres - bx, 2)
               + pow(yres - by, 2));
  }

  return 0;
}
```

„

6 Numerical

6.1 Check if float is an integer

```
#define EQ(a,b) (fabs((a) - (b)) < EPS)
#define IS_INT(a) ( EQ((a), ceil(a)) || \
                EQ((a), floor(a)) )
```

6.1.1 Big Mod

$(B^P)\%M$

```
typedef long long int lli;

long int bigmod(long long int B,
       long long int P, long long int M) {
  if (P == 0)
    return 1;
  else if (P & 1) {
    lli tmp =
       bigmod(B,(P - 1) >> 1, M) % M;
    tmp = (tmp * tmp * B) % M;
    return tmp;
  } else {
    lli tmp = bigmod(B, P >> 1, M) % M;
    return (tmp * tmp) % M;
  }
}
```

6.2 Triangle area

$$A = \frac{1}{2} * a * b * \sin(C)$$

6.3 Heron's formula

Let $s = \frac{1}{2}(a+b+c)$ then

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

6.4 Choose

$\binom{n}{k}$

```
long long memo[SIZE][SIZE]; //initialized to -1
long long binom(int n, int k){
  if (memo[n][k] != -1) return memo[n][k];
  if (n < k) return 0;
  if (n == k) return 1;
  if (k == 0) return 1;
  return memo[n][k] = binom(n - 1, k)
       + binom(n - 1, k - 1);
}
```

6.5 Modulo:

```
int mod(int a, int n) {
  return (a%n + n)%n;
}
```

6.6 LCM / GCD

$$gcd(a,b) * lcm(a,b) = a * b$$

```
int gcd(int a,int b){
  if (!b)
    return a;
  else return gcd(b, a % b);
}

struct triple{
  int gcd,x,y;
  int triple(int g = 0, int a = 0, int b = 0):
                gcd(g), x(a), y(b) {}
};

triple ExtendedEuclid(int a, int b){
  if (!b)
    return triple(a, 1, 0);

  triple t = ExtendedEuclid(b, a % b);
  return triple(t.gcd, t.y,
            t.x - (a / b) * t.y);
}

int LCM(int a, int b){
  return a * b / gcd(a, b);
}
```

6.7 Base conversion

```
void base(char *res, int num, int base){
  char tmp[100];
  int i, j;
  for (i = 0; num; i++) {
    tmp[i] =
      "0123456789ABCDEFGHIJKLM"[num % base];
    num /= base;
  }
  tmp[i] = 0;
  for (i--, j = 0; i >= 0; i--, j++)
    res[j] = tmp[i];
  res[j] = 0;
}
```

6.8 Horner's Rule

$$P(x) = \sum_{k=0}^{n} a_k x^k = a_0 + x(a_1 + x(a_2 + \cdots + (a_{n-1} + x1_n)))$$

```
double Horner(double coef[],int degree,int x) {
  double res = 0;

  for (int i = degree; i >= 0; i--)
    res = coef[i] + x * res;
  return res;
}
```

6.9 Matrix Multiplication

```
void Matrix_Multiply(int A[N][P],
            int B[P][M],int N){
  int C[N][M],i,j,k;
  for (i = 0; i < N; i++){
    for (j = 0; j < P; j++){
      C[i][j] = 0;
      for (k = 0; k < P; k++)
        C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

6.10 Long Arithmetic

Take care of leading zeroes.
Addition:

```
// make sure num1 and num2 are
// filled with '\0' after digits
void add(char *num1, char *num2, char *res){
  int i,carry=0;
  reverse(num1, num1 + strlen(num1));
  reverse(num2, num2 + strlen(num2));
  for (i = 0; num1[i] || num2[i]; i++){
    res[i] = num1[i] + num2[i]
          - '0' + carry;
    if (!num1[i] || !num2[i])
      res[i] += '0';
    if (res[i] > '9'){
      carry = 1;
      res[i] -= 10;
    } else carry = 0;
  }
  if (carry) res[i] = '1';
  reverse(res, res + strlen(res));
}
```

Multiplication

```
void mul(char *num1, char *num2, char *str) {
  int i, j, res[2*SIZE] = {0}, carry = 0;

  reverse(num1, num1 + strlen(num1));
  reverse(num2, num2 + strlen(num2));
  for (i = 0; num1[i]; i++)
    for (j = 0; num2[j]; j++)
      res[i + j] += (num1[i] - '0')
                * (num2[j] - '0');
  for (i = 2 * SIZE - 1;
      i >= 0 && !res[i]; i--);

  if (i < 0) {
    strcpy(str, "0");
    return;
  }
  for (j = 0; i >= 0; i--, j++){
    str[j] = res[i] + carry;
    carry = str[j] / 10;
    str[j] %= 10;
    str[j] += '0';
  }
  if (carry)
    str[j] = carry + '0';
}
```

6.11 Infix para Postfix

```
#define oper(a) ((a) == '+' || (a) == '-' \\
         || (a) == '*' || (a) == '/')

// true if either: !!
// b is left associative and
// its precedence is <= than a
//
// b is right associative and
// its prec is < than a
bool be_prec(char a, char b) {
  int p[300];
  p['+'] = p['-'] = 1;
  p['*'] = p['/'] = 2;
  return p[a] >= p[b];
}

string shunting_yard(string exp) {
  int i = 0;
  string res;
  stack<char> s; //operators (1 char!)

  while (i < exp.size()) {
    // if it's a function token
    // push it onto the stack

    // If it is a func arg
    // separator (e.g., a comma):
    // Until the topmost
    // elem of the stack is '('
    // pop the elem from the stack and
    // append it to res.
    // If no '(' -> error
    // do not pop '('

    if (isdigit(exp[i]) || exp[i] == 'x') {
      //number. add isalpha() for vars
      for (; i < exp.size()
           && (isdigit(exp[i])
                || exp[i] == 'x');
                i++) {
        res.push_back(exp[i]);
      }
      res.push_back('␣');
      i--; //there's a i++ down there
    } else if (exp[i] == '(') {
      s.push('(');
    } else if (exp[i] == ')') {
      while (!s.empty()
           && s.top() != '(') {
          res += s.top() + string("␣");
        s.pop();
      }
      if (s.top() != '(') ;//error
      else s.pop();
    } else if (oper(exp[i])) { //operator
```

```
      while (!s.empty()
          && oper(s.top())
          && be_prec(s.top(), exp[i])) {
        res += (s.top() + string(" "));
        s.pop();
      }
      s.push(exp[i]);
    }
    i++;
  }
  while (!s.empty()) {
    if (s.top() == '('
        || s.top() == ')')
      cout << "Error" << endl;
    res += (s.top() + string(" "));
    s.pop();
  }
  if (*(res.end() - 1) == ' ')
    res.erase(res.end() - 1);
  return res;
}
```

6.12 Calculate Postfix expression

```
// exp is in postfix
double calc(string exp) {
  stack<double> s;
  istringstream iss(exp);
  string op;

  while (iss >> op) {
  // ATTENTION TO THIS
    if (op.size() == 1 && oper(op[0])) {
      if (s.size() < 2)
        exit(-1); // error
      double a = s.top(); s.pop();
      double b  = s.top(); s.pop();
      switch (op[0]) {
        case '+': s.push(b + a); break;
        case '-': s.push(b - a); break;
        case '*': s.push(b * a); break;
        case '/': s.push(b / a); break;
      }
    } else {
      istringstream iss2(op);
      double tmp;
      iss2 >> tmp;
      s.push(tmp);
    }
  }
  return s.top();
}
```

6.13 Postfix to Infix

```
/*
 * Pass a stack with the expression
 * to rpn2infix.
 * Ex: (bottom) 3 4 5 * + (top)
 */
string rpn2infix(stack<string> &s) {
  string x = s.top();
  s.pop();
  if (isdigit(x[0])) return x;
  else return string("(") +
    rpn2infix(s) + x +
    rpn2infix(s) + string(")");
}
```

6.14 Matrix Multiplication

$$C_{ij} = \sum_{k=1}^{n} a_{ik}.b_{kj}$$

```
void matrix_mul(int A[N][P],int B[P][M]) {
  int C[N][M],i,j,k;
  for (i = 0;i<N;i++) {
    for (j = 0;j<P;j++) {
      C[i][j]=0;
      for (k=0;k<P;k++)
        C[i][j]+=A[i][k]*B[k][j];
    }
  }
}
```

6.15 Catalan Numbers

$$C_n = \frac{(2n)!}{(n+1)!n!}$$

- C_n counts the number of expressions containing n pairs of parentheses which are correctly matched
- C_n is the number of different ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.

6.16 Fibonnaci

```
long fib(long n){
  long matrix[2][2] = {{1, 1}, {1, 0}};
  long res[2][2] = {{1, 1}, {1, 0}};
  while (n) {
    if (n & 1) {
      matrix_mul(matrix, res, res);
    }
    matrix_mul(matrix, matrix, matrix);
    n /= 2;
  }
  return res[1][1];
}
```

7 Sorting / Search

7.1 Counting Sort

```
int count[SIZE] = {0};
int output[SIZE] = {0};
void linear_sort(int v[SIZE], int N) {
  int max = 0;
  for (int i = 0; i < N; ++i) {
    if (v[i] > max)
      max = v[i];
    count[v[i]]++;
  }

  for (int i = 1; i <= max; ++i) {
    count[i] += count[i-1];
  }

  for (int i = 0; i < N; ++i) {
    output[count[v[i]]-1] = v[i];
    count[v[i]]--;
  }
}
```

7.2 Binary Search - Lower bound

```
int lower_bound(int l, int r, ull q) {
  while (l < r) {
    ull mid = (l+r) / 2;

    if (v[mid] < q) {
      l = mid + 1;
    } else if (v[mid] > q) {
      r = mid - 1;
    } else {
      r = mid;
    }
  }

  return l;
}
```

7.3 Binary Search - Upper bound

```
int upper_bound(int l, int r, ull q) {
  while (l < r) {
    int mid = (l+r) / 2;

    if (v[mid] < q) {
      l = mid + 1;
    } else if (v[mid] > q) {
      r = mid - 1;
    } else {
      l = mid+1;
    }
  }

  return l;
}
```