Codings

alex4814[*]

May 10, 2014

Contents

[*] alex4814.fu@gmail.com

1 String

1.1 KMP

```
char s[MAXN], p[MAXM];
int next[MAXM];

void get_next(const char *p) {
  next[0] = -1;
  int plen = strlen(p), j = -1;
  for (int i = 1; i < plen; ++i) {
    while (j >= 0 && p[i] != p[j + 1]) { j = next[j]; }
    p[i] == p[j + 1] ? ++j : NULL;
    next[i] = j;
  }
}

// count the times string p occurs in string s
int kmp_count(const char *s, const char *p) {
  get_next(p);
  int slen = strlen(s);
  int plen = strlen(p);
  int i, j = -1, cnt = 0;
  for (i = 0; i < slen; ++i) {
    while (j >= 0 && s[i] != p[j + 1]) { j = next[j]; }
    s[i] == p[j + 1] ? ++j : NULL;
    if (j == plen - 1) { j = next[j]; ++cnt; }
  }
  return cnt;
}
```

1.2 KMP extension

1.3 Aho-Corasick Automaton

```
const int SIGMA = 26;

struct Trie {
  int chr[MAXNODE][SIGMA];  // character
  int val[MAXNODE];      // weight or value
  int size;
  Trie() { size = 1; memset(chr[0], 0, sizeof(chr[0])); }

  void insert(char *s, int w) {
    int n = strlen(s), u = 0;
    for (int i = 0; i < n; ++i) {
      int v = s[i] - 'A';
      if (!chr[u][v]) {
        memset(chr[size], 0, sizeof(chr[size]));
        val[size] = 0;
        chr[u][v] = size++;
      }
      u = chr[u][v];
    }
    val[u] = w;
  }
}

char p[MAXN][MAXN], word[MAXN];

int r, c, w;
int fail[MAXN], val[MAXN], last[MAXN];

void find(char *s) {
  int n = strlen(s), j = 0;
  for (int i = 0; i < n; ++i) {
    int c = s[i] - 'A';
    while (j && !chr[j][c]) { j = fail[j]; }
    j = chr[j][c];
    if (val[j] != 0) {
      print(i, j);
    } else if (last[j] != 0) {
      print(i, last[j]);
    }
  }
}

void get_fail() {
  queue<int> q;
  fail[0] = 0;
  for (int u = 0; u < SIGMA; ++u) {
    int v = chr[0][u];
    if (v) { f[v] = 0; q.push(v); last[v] = 0; }
  }
  while (!q.empty()) {
    int r = q.front(); q.pop();
    for (int c = 0; c < SIGMA; ++c) {
      int u = chr[r][c];
      if (!u) continue; q.push(u);
      int v = fail[r];
      while (v && !chr[v][c]) { v = f[v]; }
      fail[u] = chr[v][c];
      last[u] = val[ fail[u] ] ? fail[u] : last[ fail[u] ];
    }
  }
}
```

1.4 Suffix Array

```
int wa[MAXN], wb[MAXN], wv[MAXN], ws[MAXN];
inline int cmp(int *r, int a, int b, int l) {
  return r[a] == r[b] && r[a + l] == r[b + l];
}
// r 是待排序的串（长度已加），长度为 1 ，最大值小于n m
// 约定除 r[n - 1] 外所有的 r[i] 都大于 0, r[n - 1] = 0
void da(int *r, int *sa, int n, int m) {
  int i, j, p;
  int *x = wa, *y = wb, *t;
  for (i = 0; i < m; i++) ws[i] = 0;
  for (i = 0; i < n; i++) ws[ x[i] = r[i] ]++;
  for (i = 1; i < m; i++) ws[i] += ws[i - 1];
  for (i = n - 1; i >= 0; i--) sa[--ws[ x[i] ]] = i;
  for (j = 1, p = 1; p < n; j *= 2, m = p) {
    for (p = 0, i = n - j; i < n; i++) y[p++] = i;
    for (i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa[i] - j;
    for (i = 0; i < n; i++) wv[i] = x[y[i]];
    for (i = 0; i < m; i++) ws[i] = 0;
    for (i = 0; i < n; i++) ws[ wv[i] ]++;
    for (i = 1; i < m; i++) ws[i] += ws[i - 1];
    for (i = n - 1; i >= 0; i--) sa[--ws[ wv[i] ]] = y[i];
    for (t = x, x = y, y = t, p = 1, x[ sa[0] ] = 0, i = 1; i < n; i++) {
      x[ sa[i] ] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
    }
  }
}

int rank[MAXN], height[MAXN];
void get_height(int *r, int *sa, int n) {
   int i, j, k = 0;
   for (i = 1; i <= n; i++) rank[ sa[i] ] = i;
   for (i = 0; i < n; height[ rank[i++] ] = k)
   for (k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++);
}
```

2 Math

2.1 Cantor

```
int cantor(int *p, int n) {
  static int f[] = { 1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800 };
  int c = 0;
  for (int i = 0; i < n - 1; ++i) {
    int t = 0;
    for (int j = i + 1; j < n; ++j) {
      if (p[j] < p[i]) ++t;
    }
    c += t * f[n - 1 - i];
  }
  return c + 1;
}

int uncantor(int x, int *p, int n) {
  static int f[] = { 1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800 };
  static bool used[12] = { false };
  x -= 1;
  for (int i = 0; i < n; ++i) {
    int m = x / f[n - 1 - i];
    for (int j = 1; j <= n; ++j) {
      if (used[j]) continue;
      if (m == 0) break;
      m -= 1;
    }
    p[i] = j;
    used[j] = true;
    x %= f[n - 1 - i];
  }
}
```

2.2 Euler's Phi

```
int eulers_phi(int n) {
    int ret = 1;
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            n /= i, ret *= i - 1;
            while (n % i == 0) {
                n /= i, ret *= i;
            }
        }
    }
    return ret * (n > 1 ? n - 1 : 1);
}

void eulers_phi(int n) {
    static bool prime[MAXN];
    static int p[MAXN], _n = 0;
    static int phi[MAXN];
    for (int i = 2; i <= n; ++i) {
        if (!prime[i]) {
            phi[i] = i - 1;
            p[_n++] = i;
        }
        for (int j = 0; j < _n && p[j] * i <= n; ++j) {
            prime[ p[j] * i ] = true;
            if (i % p[j]) {
                phi[ p[j] * i ] = phi[i] * (p[j] - 1);
            } else {
                phi[ p[j] * i ] = phi[i] * (p[j] - 0); break;
            }
        }
    }
}
```

2.3 Prime

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

#define FOR(i,s,t) for (int i = s; i < t; ++i)
#define MEM(s,v) memset(s, v, sizeof(s))

#define EPS 1e-8
#define MAXN 105
#define MAXM 105
#define PI acos(-1.0)


bool is_prime(int n) {
  int x = sqrt(n);
  for (int i = 2; i <= x; ++i) {
    if (n % i == 0) { return false; }
  }
  return true;
}

int p[MAXN], np;
bool f[MAXN]; // f[i] = true means that i isn't a prime.

void get_primes(int n) {
  np = 0;
  for (int i = 2; i <= n; ++i) {
    if (is_prime(i)) { p[np++] = i; }
  }
}

/*
// O(n)
void get_primes(int n) {
  np = 0;
  for (int i = 2; i <= n; ++i) {
    if (!f[i]) { p[np++] = i; }
    for (int j = 0; j < np && p[j] * i <= n; ++j) {
      f[i * p[j]] = true;
      if (i % p[j] == 0) break;
    }
  }
}
```

```
void get_primes(int n) {
  int x = sqrt(n);
  for (int i = 2; i <= x; ++i) {
    for (int j = 2; i * j <= n; ++j) {
      f[i * j] = true;
    }
  }
  np = 0;
  for (int i = 2; i <= n; ++i) {
    if (!f[i]) { p[np++] = i; }
  }
}

*/


int main() {
  get_primes(MAXN);
  for (int i = 0; i < np; ++i) {
    printf("%d,", p[i]);
  }
  return 0;
}
```

2.4 Extended GCD

```
// solution (x, y) in integers to ax + by = gcd(a, b)
// recursive
int exgcd(int a, int b, int &x, int &y)
{
  if (b == 0) { x = 1; y = 0; return a; }
  int g = exgcd(b, a % b, y, x);
  y -= a / b * x;
  return g;
}

// solution (x, y) in integers to ax + by = gcd(a, b)
// non-recursive
int exgcd(int a, int b, int &x, int &y) {
  int g = a, v = 0, w = b; x = 1;
  while (w != 0) {
    int q = g / w, t = g % w;
    int s = x - q * v;
    x = v, g = w;
    v = s, w = t;
  }
  y = b ? (g - a * x) / b : b;
  return g;
}

// all solutions (x, y) in integers to ax + by = c
bool linear_solution(int a, int b, int c, int &x, int &y) {
  int g = exgcd(a, b, x, y);
  if (c % g != 0) return false;
  a /= g, b /= g, c /= g;
  x *= c, y *= c;
  // x = x + b * k
  // y = y - a * k;
  return true;
}
```

2.5 Factorize

```
// factorization of n
typedef pair<int, int> pii;
vector<pii> factorize(int n) {
  vector<pii> f;
  for (int i = 2; i * i <= n ; ++i) {
    if (n % i == 0) {
      int cnt = 0;
      while (n % i == 0) {
        n /= i; ++cnt;
      }
      f.push_back(make_pair(i, cnt));
    }
  }
  if (n > 1) f.push_back(make_pair(n, 1));
  return f;
}
```

2.6 XunHuanJie

```
// 求置换的循环节
// perm[n为]0..n的一个置换-1
// 返回置换最小周期，返回循环节个数num
int polya(int perm[], int n, int &num) {
  int v[MAXN] = { 0 }, ret = 1;
  int i, j, p;
  for (i = num = 0; i < n; ++i) {
    if (v[i]) continue;
    for (++num, j = 0, p = i; !v[p = perm[p]]; ++j) {
      v[p] = 1;
    }
    ret *= j / gcd(ret, j);
  }
  return ret;
}
```

3 Data Structure

3.1 Heap

```
void heapify(int *a, int n, int i) {
  int l = 2 * i, r = l + 1;
  int ix = (l <= n && a[l] > a[i]) ? l : i;
  if (r <= n && a[r] > a[ix]) ix = r;
  if (ix != i) {
    swap(a[i], a[ix]);
    heapify(a, n, ix);
  }
}

void build_heap(int *a, int n) {
  for (int i = n / 2; i >= 1; --i) {
    heapify(a, n, i);
  }
}

int heap_extract(int *a, int n) {
  if (n < 1) return -1;
  int ret = a[1];
  a[1] = a[n--];
  heapify(a, n, 1);
  return ret;
}

void heap_increase_key(int *a, int i, int key) {
  if (key < a[i]) return;
  a[i] = key;
  while (i > 1 && a[i / 2] < a[i]) {
    swap(a[i], a[i / 2]);
    i >>= 1;
  }
}

void heap_insert(int *a, int n, int key) {
  a[++n] = -INF;
  heap_increase_key(a, n, key);
}
```

3.2 Treap

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

#define FOR(i,s,t) for (int i = s; i < t; ++i)
#define MEM(s,v) memset(s, v, sizeof(s))

#define EPS 1e-8
#define _N 100005
#define _M 105
#define PI acos(-1.0)


struct Node {
  Node *ch[2];
  int key;
  int aux;
  int cnt;
  Node () {}
  Node (int key, int aux) : key(key), aux(aux) {
    ch[0] = ch[1] = NULL, cnt = 1;
  }
  bool operator < (const Node& rhs) const {
    return aux < rhs.aux;
  }
  int cmp(int x) const {
    if (x == key) return -1;
    return x < key ? 0 : 1;
  }
  void maintain() {
    cnt = 1;
    if (ch[0] != NULL) { cnt += ch[0]->cnt; }
    if (ch[1] != NULL) { cnt += ch[1]->cnt; }
  }
}*root;

Node node[_N];

void rotate(Node *&p, int d)
{
  Node *k = p->ch[d ^ 1];
  p->ch[d ^ 1] = k->ch[d]; k->ch[d] = p;
  p->maintain(); k->maintain(); p = k;
}

void insert(Node *&p, int x, int aux)
{
  if (p == NULL) {
    //p = new Node(x, aux);
    p = &node[n_cnt++];
    p->key = x, p->aux = aux, p->cnt = 1;
    p->ch[0] = p->ch[1] = NULL;
  } else {
    int d = (x < p->key ? 0 : 1); //可能有相同的点
    insert(p->ch[d], x, aux);
    if (p->ch[d]->aux > p->aux) {
      rotate(p, d ^ 1);
    }
  }
  p->maintain();
}

/*
void print(Node *p)
{
  putchar('(');
  if (p->ch[0] != NULL) print(p->ch[0]);
  printf("%s/%d", p->key.c_str(), p->aux);
  if (p->ch[1] != NULL) print(p->ch[1]);
  putchar(')');
}
*/
void print(Node *p)
{
}

int main()
{
  int key, aux;
  scanf("%d", &n);
  FOR (i, 0, n) {
    scanf("%d%d", &key, &aux);
    insert(root, key, aux);
  }
  printf("YES\n");
  print(p);
}
```

3.3 AVL

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

#define FOR(i,s,t) for (int i = s; i < t; ++i)
#define MEM(s,v) memset(s, v, sizeof(s))

#define EPS 1e-8
#define MAXN 100005
#define MAXM 105

struct AvlNode {
  AvlNode *ch[2];
  int key;
  int height;

  AvlNode(const int &key, AvlNode *lc, AvlNode *rc, int h = 0)
    :key(key), height(h) { ch[0] = lc, ch[1] = rc; }
};

inline int height(AvlNode *rt) {
  return rt == NULL ? -1 : rt->height;
}

// d = 0 for left rotate
void rotate(AvlNode *&rt, int d) {
  AvlNode *k0 = rt->ch[d ^ 1];
  rt->ch[d ^ 1] = k0->ch[d]; k0->ch[d] = rt;
  rt->height = max(height(rt->ch[d]), height(rt->ch[d ^ 1])) + 1;
  k0->height = max(height(k0->ch[d]), rt->height) + 1;
  rt = k0;
}

void insert(AvlNode *&rt, int &x) {
  if (rt == NULL) {
    rt = new AvlNode(x, NULL, NULL);
  } else if (x < rt->key) {
    insert(rt->ch[0], x);
    if (height(rt->ch[0]) - height(rt->ch[1]) == 2) {
      if (x < rt->ch[0]->key) {
        rotate(rt, 1);
      } else {
        rotate(rt->ch[0], 0);
        rotate(rt, 1);
      }
    }
  } else if (x > rt->key) {
    insert(rt->ch[1], x);
    if (height(rt->ch[1]) - height(rt->ch[0]) == 2) {
      if (x > rt->ch[1]->key) {
        rotate(rt, 0);
      } else {
        rotate(rt->ch[1], 1);
        rotate(rt, 0);
      }
    }
  }
  rt->height = max(height(rt->ch[0]), height(rt->ch[1])) + 1;
}

AvlNode *root;
int n;

int main() {
  root = NULL;
  scanf("%d", &n);
  for (int i = 0, x; i < n; ++i) {
    scanf("%d", &x);
    insert(root, x);
  }
  printf("%d\n", root->key);
  return 0;
}
```

3.4 Cartesian Tree

```
struct Node {
  string key;
  int aux;
  int parent, ch[2];
}treap[_N];

bool cmp(Node a, Node b)
{
  return a.key < b.key;
}

void init(int n)
{
  FOR (i, 0, n) { treap[i].parent = treap[i].ch[0] = treap[i].ch[1] = -1; }
}

int stack[_N];
int build(int n)
{
  int sp = -1;
  FOR (i, 0, n) {
    int k = sp;
    while (k >= 0 && treap[ stack[k] ].aux < treap[i].aux) { --k; }
    if (k != -1) {
      treap[i].parent = stack[k];
      treap[ stack[k] ].ch[1] = i;
    }
    if (k < sp) {
      treap[ stack[k + 1] ].parent = i;
      treap[i].ch[0] = stack[k + 1];
    }
    stack[sp = ++k] = i;
  }
  treap[ stack[0] ].parent = -1;
  return stack[0];
}
```

3.5 SPlay

```
/*
                An implementation of top-down splaying
                    D. Sleator <sleator@cs.cmu.edu>
                             March 1992
 */
#include <stdlib.h>
#include <stdio.h>
int size;  /* number of nodes in the tree */
           /* Not actually needed for any of the operations */
typedef struct tree_node Tree;
 struct tree_node
{
    Tree * left, * right;
    int item;
};

Tree * splay (int i, Tree * t)
{
 /* Simple top down splay, not requiring i to be in the tree t.   */
 /* What it does is described above.                              */
    Tree N, *l, *r, *y;
    if (t == NULL)
        return t;
    N.left = N.right = NULL;
    l = r = &N;
    for (;;)
    {
        if (i < t->item)
        {
            if (t->left == NULL)
            {
                break;
            }
            if (i < t->left->item)
            {
                y = t->left;                            /* rotate right */
                t->left = y->right;
                y->right = t;
                t = y;
                if (t->left == NULL)
                {
                    break;
                }
            }
            r->left = t;                                /* link right */
            r = t;
            t = t->left;
        }
        else if (i > t->item)
        {
            if (t->right == NULL)
            {
                break;
            }
            if (i > t->right->item)
            {
                y = t->right;                          /* rotate left */
                t->right = y->left;
                y->left = t;
                t = y;
                if (t->right == NULL)
                {
                    break;
                }
            }
            l->right = t;                              /* link left */
            l = t;
            t = t->right;
        }
        else
        {
            break;
        }
    }
    l->right = t->left;                                /* assemble */
    r->left = t->right;
    t->left = N.right;
    t->right = N.left;
    return t;
}
 /* Here is how sedgewick would have written this.                    */
/* It does the same thing.                                           */
Tree * sedgewickized_splay (int i, Tree * t)
{
    Tree N, *l, *r, *y;
    if (t == NULL)
    {
        return t;
    }
    N.left = N.right = NULL;
    l = r = &N;
    for (;;)
    {
        if (i < t->item)
        {
            if (t->left != NULL && i < t->left->item)
            {
                y = t->left;
                t->left = y->right;
                y->right = t;
                t = y;
            }
            if (t->left == NULL)
            {
                break;
            }
            r->left = t;
            r = t;
            t = t->left;
        }
        else if (i > t->item)
        {
            if (t->right != NULL && i > t->right->item)
            {
                y = t->right;
                t->right = y->left;
                y->left = t;
                t = y;
            }
            if (t->right == NULL)
            {
                break;
            }
            l->right = t;
            l = t;
            t = t->right;
        }
        else
        {
            break;
        }
    }
    l->right=t->left;
    r->left=t->right;
    t->left=N.right;
    t->right=N.left;
    return t;
}

Tree * insert(int i, Tree * t)
{
/* Insert i into the tree t, unless it's already there.    */
/* Return a pointer to the resulting tree.                 */
    Tree * new;

    new = (Tree *) malloc (sizeof (Tree));
    if (new == NULL)
    {
        printf("Ran␣out␣of␣space\n");
        exit(1);
    }
    new->item = i;
    if (t == NULL)
    {
        new->left = new->right = NULL;
        size = 1;
        return new;
    }
    t = splay(i,t);
    if (i < t->item)
    {
        new->left = t->left;
        new->right = t;
        t->left = NULL;
        size ++;
        return new;
    }
    else if (i > t->item)
    {
        new->right = t->right;
        new->left = t;
        t->right = NULL;
        size++;
        return new;
    }
    else
    {
        /* We get here if it's already in the tree */
        /* Don't add it again                      */
        free(new);
        return t;
    }
}
```

```
Tree * delete(int i, Tree * t)
{
/* Deletes i from the tree if it's there.               */
/* Return a pointer to the resulting tree.              */
    Tree * x;
    if (t==NULL)
    {
        return NULL;
    }
    t = splay(i,t);
    if (i == t->item)
    {               /* found it */
        if (t->left == NULL)
        {
            x = t->right;
        }
        else
        {
            x = splay(i, t->left);
            x->right = t->right;
        }
        size--;
        free(t);
        return x;
    }
    return t;                         /* It wasn't there */
}

int main(int argv, char *argc[])
{
/* A sample use of these functions.  Start with the empty tree,         */
/* insert some stuff into it, and then delete it                        */
    Tree * root;
    int i;
    root = NULL;              /* the empty tree */
    size = 0;
    for (i = 0; i < 1024; i++)
    {
        root = insert((541*i) & (1023), root);
    }
    printf("size␣=␣%d\n", size);
    for (i = 0; i < 1024; i++)
    {
        root = delete((541*i) & (1023), root);
    }
    printf("size␣=␣%d\n", size);
}
```

4 Graph

4.1 2-Sat

```
struct TwoSat {
    int n, c;
    vector<int> g[MAXV << 1];
    bool mark[MAXV << 1];
    int s[MAXV << 1];

    bool dfs(int x) {
        if (mark[x ^ 1]) return false;
        if (mark[x]) return true;

        mark[x] = true;
        s[c++] = x;
        for (int i = 0; i < g[x].size(); ++i) {
            if (!dfs(g[x][i])) return false;
        }
    }

    void init(int n) {
        this->n = n;
        for (int i = 0; i < n * 2; ++i) {
            g[i].clear();
        }
        memset(mark, 0, sizeof(mark));
    }

    // x = xval or y = yval
    void add_clause(int x, int xval, int y, int yval) {
        x = x * 2 + xval;
        y = y * 2 + yval;
        g[x ^ 1].push_back(y);
        g[y ^ 1].push_back(x);
    }

    bool solve() {
        for (int i = 0; i < n * 2; i += 2) {
            if (!mark[i] && !mark[i + 1]) {
                c = 0;
                if (!dfs(i)) {
                    while (c > 0) { mark[ s[--c] ] = false; }
                    if (!dfs(i + 1)) return false;
                }
            }
        }
        return true;
    }
};
```

4.2 Path, Flow

```
// 最大流

// 1.SAP(Shortest Augmenting Path)
//   EK(Edmonds Karp)
int cap[MAXN][MAXN];
int flow[MAXN][MAXN];
int p[MAXN], rnet[MAXN];

int max_flow(int s, int t)
{
  int f = 0;
  queue<int> q;
  memset(flow, 0, sizeof(flow));
  while (1) {
    memset(rnet, 0, sizeof(rnet));
    rnet[s] = INT_MAX;
    q.push(s);
    while (!q.empty()) {
      int u = q.front(); q.pop();
      for (int v = 0; v <= n + 1; ++v) {
        if (!rnet[v] && cap[u][v] > flow[u][v]) {
          p[v] = u; q.push(v);
          rnet[v] = min(rnet[u], cap[u][v] - flow[u][v]);
        }
      }
    }
    if (rnet[t] == 0) break;
    for (int u = t; u != s; u = p[u]) {
      flow[p[u]][u] += rnet[t];
      flow[u][p[u]] -= rnet[t];
    }
    f += rnet[t];
  }
  return f;
}


// 邻接表
int first[MAXN], next[MAXN];
int u[MAXN], v[MAXN], w[MAXN];
void read_graph(int n, int m)
{
  scanf("%d%d", &n, &m);
  for (int i = 0; i < n; ++i) first[i] = -1;
  for (int e = 0; e < m; ++e) {
    scanf("%d%d%d", &u[e], &v[e], &w[e]);
    next[e] = first[u[e]];
    first[u[e]] = e;
  }
}

/** 单源最短路dijkstra **/

// O(n^2) 邻接矩阵
int vis[MAXN], d[MAXN];
void dijkstra(int s)
{
  memset(vis, 0, sizeof(vis));
  for (int i = 0; i < n; ++i) {
    d[i] = (i == s) ? 0 : INF;
  }
  for (int i = 0; i < n; ++i) {
    int x, m = INF;
    for (int y = 0; y < n; ++y) {
      if (!vis[y] && d[y] <= m) {
        m = d[x = y];
      }
      vis[x] = 1;
    }
    for (int y = 0; y < n; ++y) {
      if (d[y] < d[x] + w[x][y]) {
        d[y] = d[x] + w[x][y];
        // fa[y] = x; 记录路径//
      }
    }
  }
}
// O(nm) 邻接表
void dijkstra(int s)
{
  memset(vis, 0, sizeof(vis));
  for (int i = 0; i < n; ++i) {
    d[i] = (i == s) ? 0 : INF;
  }
  for (int i = 0; i < n; ++i) {
    int x, m = INF;
    for (int e = first[i]; e != -1; e = next[e]) {
      if (!vis[v[e]] && w[e] <= m) {
        m = d[x = v[e]];
      }
      vis[x] = 1;
    }
    for (int e = first[x]; e != -1; e = next[e]) {
      if (d[v[e]] < d[x] + w[e]) {
        d[v[e]] = d[x] + w[e];
        fa[v[e]] = x;
      }
    }
  }
}
```

```cpp
// 优先队列
struct cmp {
  bool operator() (const int a, const int b) {
    return a % 10 > b % 10;
  }
};
typedef pair<int, int> pii;
priority_queue< pii, vector<pii>, greater<pii> > q;

bool done[MAXN];
void dijkstra(int s)
{
  for (int i = 0; i < n; ++i) {
    d[i] = (i == s) ? 0 : INF;
  }
  memset(done, 0, sizeof(done));
  q.push(make_pair(d[s], 0));
  while (!q.empty()) {
    pii u = q.top(); q.pop();
    int x = u.second;
    if (done[x]) continue;
    done[x] = true;
    for (int e = first[x]; e != -1; e = next[e]) {
      d[v[e]] = d[x] + w[e];
      q.push(make_pair(d[v[e]], v[e]));
    }
  }
}

/** Bellman Ford **/

// O(nm)
void bellman_ford(int s)
{
  for (int i = 0; i < n; ++i) {
    d[i] = INF;
  }
  d[0] = 0;
  for (int k = 0; k < n - 1; ++k) {
    for (int i = 0; i < m; ++i) {
      int x = u[i], y = v[i];
      if (d[x] < INF) {
        if (d[y] < d[x] + w[i]) {
          d[y] = d[x] + w[i];
        }
      }
    }
  }
}

void bellman_ford(int s)
{
  for (int i = 0; i < n; ++i) {
    d[i] = (i == s) ? 0 : INF;
  }
  bool inq[MAXN];
  memset(inq, 0, sizeof(inq));
  queue<int> q;
  q.push(s);
  while (!q.empty()) {
    int x = q.front(); q.pop();
    inq[x] = false;
    for (int e = first[x]; e != -1; e = next[e]) {
      if (d[v[e]] > d[x] + w[e]) {
        d[v[e]] = d[x] + w[e];
        if (!inq[v[e]]) {
          inq[v[e]] = true;
          q.push(v[e]);
        }
      }
    }
  }
}
```

4.3 Bipartite Match

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
using namespace std;

#define FOR(i,s,t) for (int i = s; i < t; ++i)
#define MEM(s, v) memset(s, v, sizeof(s))
#define EPS 1e-8
#define _N 100005
#define _M _N
#define PI acos(-1.0)

int g[_N][_M];
int limits[_M];
int vis[_M];
int match[_M];

bool dfs(int u, int m)
{
  FOR (v, 0, m) {
    if (g[u][v] && !vis[v]) {
      vis[v] = true;
      if (match[v] == -1 || dfs(match[v], m)) {
        match[v] = u;
        return true;
      }
    }
  }
  return false;
}

int bipartite_match(int n, int m)
{
  int cnt = 0;
  MEM (match, -1);
  FOR (i, 0, n) {
    MEM (vis, false);
    if (dfs(i, m)) ++cnt;
  }
  return cnt;
}

int main()
{

  return 0;
}
```

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <iostream>
#include <string>
#include <algorithm>
```

```
using namespace std;

#define FOR(i,s,t) for (int i = s; i < t; ++i)
#define MEM(s,v) memset(s, v, sizeof(s))

#define EPS 1e-8
#define _N 100005
#define _M 105
#define PI acos(-1.0)

int n;
int g[_N][_N], s[_N][_N], e[_N][_N];
int mx[_N], my[_N];
int lx[_N], ly[_N];
int q[_N], qf, qb;
int py[_N];

bool match(int r)
{
  while (true) {
    MEM (py, -1);
    for (qf = 0, qb = 1, q[0] = r; qf < qb; ) {
      for (int x = q[qf++], y = 0; y < n; ++y) {
        if (lx[x] + ly[y] == g[x][y] && py[y] == -1) {
          q[qb++] = my[y]; py[y] = x;
          if (my[y] == -1) {
            for (int ty = 0; ty != -1; y = ty) {
              ty = mx[x = py[y]], my[y] = x, mx[x] = y;
            }
            return true;
          }
        }
      }
    }
    int d = INF;
    FOR (i, 0, qb) FOR (y, 0, n) if (py[y] == -1) {
      if (g[ q[i] ][y] != INF) {
        d = min(d, lx[ q[i] ] + ly[y] - g[ q[i] ][y]);
      }
    }
    if (d == INF) break;

    FOR (i, 0, qb) { lx[ q[i] ] -= d; }
    FOR (y, 0, n) if (py[y] != -1) { ly[y] += d; }
  }
  return false;
}

int kuhn_munkres()
{
  MEM (mx, -1); MEM (my, -1);
  MEM (lx, 0); MEM (ly, 0);
  FOR (x, 0, n) FOR (y, 0, n) { lx[x] = max(lx[x], g[x][y]); }
  FOR (x, 0, n) if (!match(x)) { mx[x] = -1; }

  int cost = 0;
  FOR (x, 0, n) if (mx[x] != -1) { cost += g[x][ mx[x] ]; }
  return cost;
}

int main()
{

  return 0;
}
```

```
#include <cstdio>
#include <cstdlib>
```

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

#define FOR(i,s,t) for (int i = s; i < t; ++i)
#define MEM(s, v) memset(s, v, sizeof(s))
#define EPS 1e-8
#define _N 40
#define _M 40
#define MAXN 100005
#define PI acos(-1.0)

int g[_N][_M];
int limits[_M];
int vis[_M];
int match[_M][MAXN];
int n_match[_M];

bool dfs(int u, int m)
{
  FOR (v, 0, m) {
    if (g[u][v] && !vis[v]) {
      vis[v] = true;
      if (n_match[v] < limits[v]) {
        match[v][ n_match[v]++ ] = u;
        return true;
      } else {
        FOR (i, 0, n_match[v]) {
          if (dfs(match[v][i], m)) {
            match[v][i] = u;
            return true;
          }
        }
      }
    }
  }
  return false;
}

int bipartite_multi_match(int n, int m)
{
  int cnt = 0;
  MEM (n_match, 0);
  FOR (i, 0, n) {
    MEM (vis, false);
    if (dfs(i, m)) ++cnt;
  }
  return cnt;
}


int main()
{

  return 0;
}
```

5 Geometry

5.1 2-dimension

5.1.1 Structures

```
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;

#define MAXN 100005

const double EPS = 1e-8;
const double PI = M_PI;

inline int sgn(double x) {
    return (x > EPS) - (x < -EPS);
}

typedef struct Point {
  double x, y;
  Point(double x = 0, double y = 0): x(x), y(y) {}
} Vector;

struct Segment {
  Point a, b;
  Segment(Point a, Point b): a(a), b(b) {}
};

struct Line {
  Point p;
  Vector v;
  double a;
  Line(Point p = Point(), Vector v = Vector(1, 0)): p(p), v(v) { a = atan2(v.y, v.x); }
  bool operator < (const Line &b) const { return a < b.a; }
};

struct Circle {
  Point c;
  double r;
  Circle(Point c = Point(), double r = 0): c(c), r(r) {}
};
```

5.1.2 Point & Vector

```
Vector operator + (Vector a, Vector b) {
  return Vector(a.x + b.x, a.y + b.y);
}
Vector operator - (Vector a, Vector b) {
  return Vector(a.x - b.x, a.y - b.y);
}
Vector operator * (Vector a, double k) {
  return Vector(a.x * k, a.y * k);
}
Vector operator / (Vector a, double k) {
  return Vector(a.x / k, a.y / k);
}
bool operator == (const Point &a, const Point &b) {
  return sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0;
}
bool operator < (const Point &a, const Point &b) {
  return a.y < b.y || (a.y == b.y && a.x < b.x);
}

inline double dot(Vector a, Vector b) {
  return a.x * b.x + a.y * b.y;
}
inline double cross(Vector a, Vector b) {
  return a.x * b.y - a.y * b.x;
}
inline double xmult(Point a, Point b, Point c) {
  return cross(b - a, c - a);
}
```

```
inline double length(Vector a) {
  return sqrt(dot(a, a));
}
inline double angle(Vector a, Vector b) {
  return acos(dot(a, b) / length(a) / length(b));
}
inline Vector rotate(Vector a, double rad) {
  return Vector(a.x * cos(rad) - a.y * sin(rad), a.x * sin(rad) + a.y * cos(rad));
}
```

5.1.3 Distance

```
inline double dis_pp(Point a, Point b) {
  return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
inline double dis_pl(Point p, Point a, Point b) {
  Vector v1 = b - a, v2 = p - a;
  return fabs(cross(v1, v2)) / length(v1);
}
inline double dis_ps(Point p, Point a, Point b) {
  Vector v1 = b - a, v2 = p - a, v3 = p - b;
  if (a == b) return length(p - a);
  if (sgn(dot(v1, v2)) < 0) return length(v2);
  if (sgn(dot(v1, v3)) > 0) return length(v3);
  return fabs(cross(v1, v2)) / length(v1);
}
```

5.1.4 Position

```
// change < to <=, if improper
inline bool is_proper_intersection_ss(Point a1, Point a2, Point b1, Point b2) {
  double d1 = cross(a2 - a1, b1 - a1), d2 = cross(a2 - a1, b2 - a1);
  double d3 = cross(b2 - b1, a1 - b1), d4 = cross(b2 - b1, a2 - b1);
  return sgn(d1) * sgn(d2) < 0 && sgn(d3) * sgn(d4) < 0;
}
inline bool is_proper_intersection_ls(Point a1, Point a2, Point b1, Point b2) {
  double d1 = cross(a2 - a1, b1 - a1), d2 = cross(a2 - a1, b2 - a1);
  return sgn(d1) * sgn(d2) < 0;
}
inline bool is_on_ps(Point p, Point a, Point b) {
  return sgn(cross(a - p, b - p)) == 0 && sgn(dot(a - p, b - p)) < 0;
}
inline bool is_on_pl(Point p, Point a, Point b) {
  return sgn(cross(a - p, b - p)) == 0;
}
inline bool is_parallel_ll(Point a1, Point a2, Point b1, Point b2) {
  return sgn(cross(a2 - a1, b2 - b1)) == 0;
}
inline bool is_left_pl(Point p, Line l) {
  return cross(l.v, p - l.p) > 0;
}
// p[] for polygon, p[n] = p[0]
// function: whether point a is inside or on the boundry of polygon p
int is_in_pp(Point a, Point *p, int n) {
  int wn = 0;
  for (int i = 0; i < n; ++i) {
    if (is_on_ps(a, p[i], p[i + 1])) return -1;     // on edge;
    int k = sgn(xmult(p[i], p[i + 1], a));
    int d1 = sgn(p[i].y - a.y);
    int d2 = sgn(p[i + 1].y - a.y);
    if (k > 0 && d1 <= 0 && d2 > 0) ++wn;
    if (k < 0 && d2 <= 0 && d1 > 0) --wn;
  }
  return wn != 0;
}

// -1 for d > r, 0 for d == r, 1 for d < r
inline int relative_position_cl(Circle c, Point p, Vector v) {
  double d = dis_pl(c.c, p, p + v);
  return sgn(c.r - d);
}
// -1 for outside, 0 for on, 1 for inside
inline int relative_position_cp(Circle c, Point p) {
  double d = dis_pp(c.c, p);
  return sgn(c.r - d);
}
```

5.1.5 Intersection

```
Point intersection_ll(Point a, Vector i, Point b, Vector j) {
  Vector k = a - b;
  double t = cross(j, k) / cross(i, j);
  return a + i * t;
}
Point intersection_ll(Line a, Line b) {
  Vector u = a.p - b.p;
  double t = cross(b.v, u) / cross(a.v, b.v);
  return a.p + a.v * t;
}
Point projection_pl(Point p, Point a, Point b) {
  Vector v = b - a;
  return a + v * (dot(v, p - a) / dot(v, v));
}
```

5.1.6 Convex Hull

```
// p[] for original points in polygon(sorted)
// ch[] for final points in convex hull
// return nums of points
// ch[ix] = ch[0]
// <= 0 for all points in a line, < otherwise
int convex_hull(Point ch[], Point p[], int n)
{
  sort(p, p + n);
  int ix = 0;
  for (int i = 0; i < n; ++i) {
    while (ix > 1 && xmult(ch[ix - 2], ch[ix - 1], p[i]) < 0) --ix;
    ch[ix++] = p[i];
  }
  int t = ix;
  for (int i = n - 2; i >= 0; --i) {
    while (ix > t && xmult(ch[ix - 2], ch[ix - 1], p[i]) < 0) --ix;
    ch[ix++] = p[i];
  }
  return n > 1 ? ix - 1 : ix;
}
```

5.1.7 Area

```
inline double area_of_triangle(Point a, Point b, Point c) {
    return length(cross(b - a, c - a)) / 2;
}
```

5.1.8 Circle

```
// 计算直线与圆的交点保证直线与圆有交点，
// 计算线段与圆的交点可用这个函数后判点是否在线段上
void intersection_cl(Circle c, Point p, Vector v, Point &p1, Point &p2) {
  Point l1 = p, l2 = p + v;
  Vector u = Vector(-v.y, v.x);
  Point p0 = intersection_ll(p, v, c.c, u);
  double d1 = dis_pp(p0, c.c);
  double d2 = dis_pp(l1, l2);
  double t = sqrt(c.r * c.r - d1 * d1)/ d2;
  p1.x = p0.x + (l2.x - l1.x) * t;
  p1.y = p0.y + (l2.y - l1.y) * t;
  p2.x = p0.x - (l2.x - l1.x) * t;
  p2.y = p0.y - (l2.y - l1.y) * t;
}

// 前提: 保证圆与圆有交点，圆心不重合
void intersection_cc(Circle c1, Circle c2, Point& p1, Point& p2){
  double d = dis_pp(c1.c, c2.c);
  double t = (1.0 + (c1.r * c1.r - c2.r * c2.r) / d / d) / 2;
  Point u = Point(c1.c.x + (c2.c.x - c1.c.x) * t, c1.c.y + (c2.c.y - c1.c.y) * t);
  Point v = Point(u.x + c1.c.y - c2.c.y, u.y - c1.c.x + c2.c.x);
  intersection_cl(c1, u, v - u, p1, p2);
}

// 计算圆外一点与圆的两个切点
void point_of_tangency_cp(Circle c, Point p, Point &p1, Point &p2) {
  double d = dis_pp(c.c, p);
  double theta = asin(c.r / d);
  Vector v1 = rotate(c.c - p, theta);
  Vector v2 = rotate(c.c - p, 2 * PI - theta);
  p1 = p + v1 / length(v1) * d * cos(theta);
  p2 = p + v2 / length(v2) * d * cos(theta);
}

// 计算最小的圆覆盖平面上的点集
Circle min_circle_cover(Point *p, int n) {
  Point c = p[0]; double r = 0;
  for (int i = 1; i < n; ++i) {
    if (sgn(dis_pp(c, p[i]) - r) <= 0) continue;
    c = p[i], r = 0;
    for (int j = 0; j < i; ++j) {
      if (sgn(dis_pp(c, p[j]) - r) <= 0) continue;
      c.x = (p[i].x + p[j].x) / 2;
      c.y = (p[i].y + p[j].y) / 2;
      r = dis_pp(c, p[j]);
      for (int k = 0; k < j; ++k) {
        if (sgn(dis_pp(c, p[k]) - r) <= 0) continue;
        c = circumcenter(p[i], p[j], p[k]);
        r = dis_pp(c, p[k]);
      }
    }
  }
  return Circle(c, r);
}
```

5.1.9 Segment

```
int seg_union(Segment *s, int &n) {
  int m = 0;
  for (int i = 1; i < n; ++i) {
    if (s[m].b.x < s[i].a.x) {
      s[++m] = s[i];
    } else {
      s[m].a.x = min(s[m].a.x, s[i].a.x);
      s[m].b.x = max(s[m].b.x, s[i].b.x);
    }
  }
  return n = m + 1;
}
```

5.1.10 Symmetry

```
Point symmetry_pl(Point p, Point a, Point b) {
  Vector v1 = b - a, v2 = Vector(-v1.y, v1.x);
  Point m = intersection_ll(a, v1, p, v2);
  return p + (m - p) * 2;
}
```

5.1.11 Triangle

```
Point circumcenter(Point a, Point b, Point c) {
  double x1 = b.x - a.x, y1 = b.y - a.y, e1 = (x1 * x1 + y1 * y1) / 2;
  double x2 = c.x - a.x, y2 = c.y - a.y, e2 = (x2 * x2 + y2 * y2) / 2;
  double _d = x1 * y2 - x2 * y1;
  double _x = a.x + (e1 * y2 - e2 * y1) / _d;
  double _y = a.y + (x1 * e2 - x2 * e1) / _d;
  return Point(_x, _y);
}
```

5.2 3-dimension

5.2.1 Structures

```
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;

const double EPS = 1e-8;
const double PI = acos(-1.0);

inline int sgn(double x) {
  return (x > EPS) - (x < -EPS);
}

typedef struct Point3 {
  double x, y, z;
  Point3(double x = 0, double y = 0, double z = 0): x(x), y(y), z(z) {}
} Vector3 ;
```

5.2.2 Point & Vector

```
Vector3 operator + (Vector3 a, Vector3 b) {
  return Vector3(a.x + b.x, a.y + b.y, a.z + b.z);
}
Vector3 operator - (Point3 a, Point3 b) {
  return Vector3(a.x - b.x, a.y - b.y, a.z - b.z);
}
Vector3 operator * (Vector3 a, double k) {
  return Vector3(a.x * k, a.y * k, a.z * k);
}
Vector3 operator / (Vector3 a, double k) {
  return Vector3(a.x / k, a.y / k, a.z / k);
}
bool operator == (const Point3 &a, const Point3 &b) {
  return sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0 && sgn(a.z - b.z) == 0;
}
inline double dot(Vector3 a, Vector3 b) {
  return a.x * b.x + a.y * b.y + a.z * b.z;
}
inline double length(Vector3 a) {
  return sqrt(dot(a, a));
}
inline double angle(Vector3 a, Vector3 b) {
  return acos(dot(a, b) / length(a) / length(b));
}
inline Vector3 cross(Vector3 a, Vector3 b) {
  return Vector3(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x);
}
```

5.2.3 Distance

```
inline double dis_pp(Point3 a, Point3 b) {
  return length(a - b);
}
inline double dis_pf(Point3 p, Point3 p0, Vector3 n) {
  return fabs(dot(p - p0, n));
}
```

```
inline Point3 projection_pf(Point3 p, Point3 p0, Vector3 n) {
  return p - n * dot(p - p0, n);
}
inline Point3 intersection_lf(Point3 a, Point3 b, Point3 p0, Vector3 n) {
  Vector3 v = b - a;
  double t = dot(n, p0 - a) / dot(n, b - a);
  return a + v * t;
}

inline double dis_pl(Point3 p, Point3 a, Point3 b) {
  Vector3 v1 = b - a, v2 = p - a;
  return length(cross(v1, v2)) / length(v1);
}
inline double dis_ps(Point3 p, Point3 a, Point3 b) {
  if (a == b) return length(p - a);
  Vector3 v1 = b - a, v2 = p - a, v3 = p - b;
  if (sgn(dot(v1, v2)) < 0) return length(v2);
  if (sgn(dot(v1, v3)) > 0) return length(v3);
  return length(cross(v1, v2)) / length(v1);
}
```

5.2.4 Convex Hull

```
struct Face {
  int v[3];
  Vector3 normal(Point3 *p) const {
    return cross(p[v[1]] - p[v[0]], p[v[2]] - p[v[0]]);
  }
  bool cansee(Point3 *p, int i) const {
    return dot(p[i] - p[v[0]], normal(p)) > 0;
  }
};

vector<Face> convex_hull3(Point3 *p, int n) {
  static int vis[MAXN][MAXN];
  vector<Face> ch;
  ch.push_back((Face){{ 0, 1, 2 }});
  ch.push_back((Face){{ 2, 1, 0 }});
  for (int i = 3; i < n; ++i) {
    vector<Face> next;
    // calculate the left visibility of each edge
    for (int j = 0; j < ch.size(); ++j) {
      Face &f = ch[j];
      int ret = f.cansee(p, i);
      if (!ret) next.push_back(f);
      for (int k = 0; k < 3; ++k) {
        vis[ f.v[k] ][ f.v[(k + 1) % 3] ] = ret;
      }
    }
    for (int j = 0; j < ch.size(); ++j) {
      for (int k = 0; k < 3; ++k) {
        int a = ch[j].v[k], b = ch[j].v[(k + 1) % 3];
        if (vis[a][b] != vis[b][a] && vis[a][b]) {  // (a, b)'s left is visible to p[i]
          next.push_back((Face){{ a, b, i }});
        }
      }
    }
    ch = next;
  }
  return ch;
}
```

6 Others

6.1 Mapping

```
void mapping(int a[], int n) {
    vector<pii> _m;
    for (int i = 0; i < n; ++i) {
        _m.push_back(make_pair(a[i], i));
    }
    sort(_m.begin(), _m.end());

    int _t = 0;
    for (int i = 0; i < n; ++i) {
        a[ _m[i].second ] = _t;
        if (i + 1 < n && _m[i].first != _m[i + 1].first) {
            ++_t;
        }
    }
}
```

6.2 RMQ

```
// f[i][j] 代表从 i 开始，长为 2^j 的区间中的最小（大）值
int f[MAXN][20];
// 预处理 O(nlogn)
void rmq_init(int *a, int n) {
  for (int i = 0; i < n; ++i) {
    f[i][0] = a[i];
  }
  for (int j = 1; (1 << j) <= n; ++j) {
    for (int i = 0; i + (1 << j) <= n; ++i) {
      f[i][j] = min(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
    }
  }
}
// 查询区间 [l，r] 的最小（大）值
int rmq_query(int l, int r) {
  int k = 0;
  while ((1 << (k + 1)) <= r - l + 1) ++k;
  return min(f[l][k], f[r - (1 << k) + 1][k]);
}
```

6.3 Binary Search

```
int bin_search(int *a, int l, int r, int x) {
  while (l < r) {
    int m = l + (r - l) / 2;
    if (a[m] == x) return m;
    a[m] > x ? r = m : l = m + 1;
  }
  return -1;
}

int lower_bound(int *a, int l, int r, int x) {
  while (l < r) {
    int m = l + (r - l) / 2;
    a[m] >= x ? r = m : l = m + 1;
  }
  return l;
}

int upper_bound(int *a, int l, int r, int x) {
  while (l < r) {
    int m = l + (r - l) / 2;
    a[m] <= x ? l = m + 1 : r = m;
  }
  return r;
}
```