Cheat Sheet ICPC

Contents

Cheat Sheet ICPC

Versi Satu Titik Nol

1 Dijkstra

```
int main () {
    int n,m;
    cin >> n >> m;
    vector<pii> adj[10005];
    for (int i=1;i<=m;i++) {
    int x,y,w;
    cin >> x >> y >> w;
    adj[x].pb(mp(w,y));
    }
    int s,e;
    cin >> s >> e;
    priority_queue<pii,vector<pii>,greater<pii> > d;
    int vis[10005];
    for (int i=1;i<=n;i++) vis[i]=INT_MAX;
    vis[s]=0;
    d.push(mp(0,s));
    while (!d.empty()) {
    pii cur=d.top();
    d.pop();
    for (int i=0;i<adj[cur.se].size();i++) {
      pii next=adj[cur.se][i];
      if (vis[cur.se]+next.fi<vis[next.se]) {
        vis[next.se]=vis[cur.se]+next.fi;
        d.push(mp(vis[next.se],next.se));
      }
    }
    }
    if (vis[e]==INT_MAX) cout << "NO" << endl; else cout <<
        ↪ vis[e] << endl;
    return 0;
}
```

2 MST

```
int main () {
    int n,m;
    cin >> n>> m;
    vector<pii> adj[10005];
    for (int i=1;i<=m;i++) {
      int u,v,w;
      cin >> u >> v >> w;
      adj[u].pb(mp(w,v));
      adj[v].pb(mp(w,u));
    }
    bool MST[10005];
    memset (MST,0,sizeof(MST));
    int vis[10005];
    for (int i=1;i<=n;i++) vis[i]=INT_MAX;
    priority_queue<pii,vector<pii>,greater<pii> > pq;
    pq.push(mp(0,1));
    vis[1]=0;
    while (!pq.empty()) {
      pii cur=pq.top();
      pq.pop();
      MST[cur.se]=1;
      for (int i=0;i<adj[cur.se].size();i++) {
        pii next=adj[cur.se][i];
        if (!MST[next.se]&&vis[next.se]>next.fi) {
          vis[next.se]=next.fi;
          pq.push(next);
        }
      }
    }
    for (int i=1;i<=n;i++) cout << vis[i] << "␣";
    return 0;
}
```

3 Knapsack

```
int n,v[n+5],w[n+5],cap,dp[n+5];

int solve(int posi,int rem) {
  if (posi==0||rem==0) return 0;
  if (dp[posi][rem]!=-1) return dp[posi][rem];
  int ret=solve(posi-1,rem);
  if (rem>=w[posi]) ret=max(ret,solve(posi-1,rem-w[posi])+v
      ↪ [posi]);
  dp[posi][rem]=ret;
  return ret;
}

int main () {
  cin >> n;
  for (int i=1;i<=n;i++) cin >> v[i];
  for (int i=1;i<=n;i++) cin >> w[i];
  memset (dp,-1,sizeof(dp));
  cout << solve(n,cap) << endl;
  return 0;
}
```

4 Coin Change

```
int n,m,c[55],dp[55][255];

int solve(int pos,int rem) {
    if (rem==0) return 1;
    if (pos<1||rem<0) return 0;
    if (dp[pos][rem]!=-1) return dp[pos][rem];
    return dp[pos][rem]=solve(pos,rem-c[pos])+solve(pos-1,
        ↪ rem);
}

int main () {
    cin >> n >> m;
    for (int i=1;i<=m;i++) cin >> c[i];
    memset (dp,-1,sizeof(dp));
    cout << solve(m,n);
    return 0;
}
```

5 Negative Cycle Detection

```
int main () {
  int n,m;
  cin >> n >> m;
  vector<pair<pair<int,int>,int> > edge;
  for (int i=1;i<=m;i++) {
    int u,v,w;
    cin >> u >> v >> w;
    edge.pb(mp(mp(u,v),w));
  }
  int dist[35];
  for (int i=1;i<=n;i++) dist[i]=123456789;
  bool y=0;
  dist[1]=0;
  for (int i=1;i<n;i++) {
    for (int j=0;j<m;j++) {
      if (dist[edge[j].fi.se]>dist[edge[j].fi.fi]+edge[j].
          ↪ se) {
        dist[edge[j].fi.se]=dist[edge[j].fi.fi]+edge[j].se;
      }
    }
  }
  for (int i=0;i<m;i++) {
    if (dist[edge[i].fi.se]>dist[edge[i].fi.fi]+edge[i].se)
        ↪ {
      y=1;
    }
  }
```

```
  if (y) cout << "ada␣negative␣cycle" << endl;
  return 0;
}
```

6 Toposort

```
stack<int> ans;
int vis[105];
bool cycle=0;
vector<int> adj[105];

void toposort(int x) {
  vis[x]=1;
  for (int j=0;j<adj[x].size();j++) {
    if (vis[adj[x][j]]==0) {
      vis[adj[x][j]]=1;
      toposort(adj[x][j]);
    } else if (vis[adj[x][j]]==1) cycle=1;
  }
  ans.push(x);
  vis[x]=2;
}

int main () {
    int n,m;
    cin >> n >> m;
    for (int i=1;i<=m;i++) {
      int x,y;
      cin >> x >> y;
      adj[x].pb(y);
    }
    memset (vis,0,sizeof(vis));
    for (int i=1;i<=n;i++) {
      if (!vis[i]) toposort(i);
    }

    if (cycle) cout << "Ada␣Cycle" << endl;
    else {
      while (!ans.empty()) {
        cout << ans.top() <<"␣";
        ans.pop();
      }
    }
    return 0;
}
```

7 a/b mod m

```
LL gcd(LL a,LL b, LL &x, LL &y) {
  if (a==0) {
    x=0;
    y=1;
    return b;
  }
  LL x1,y1;
  LL d=gcd(b%a,a,x1,y1);
  x=y1-(b/a)*x1;
  y=x1;
  return d;
}

LL inverse_modulo(LL a,LL m) {
  LL x,y;
  LL ans=gcd(a,m,x,y);
  return (x%m + m)%m;
}

int main () {
    LL n,k,m;
    cin >> n >> k >> m;
    LL up=1;
    for (LL i=n;i>n-k;i--) {
      up*=i;
      up%=m;
    }
    //cout << up << endl;
    LL down=1;
    for (LL i=1;i<=k;i++) {
      down*=i;
      down%=m;
    }
    //cout << down << endl;
    LL im=inverse_modulo(down,m);
    if (im<0) im+=m;
    //cout << im << endl;
    cout << ((up%m)*(im%m))%m << endl;
    return 0;
}
```

8 Sieve of Erastothenes

```
bitset<100000005> prima;

void sieve(int n) {
  prima.set(); //ubah semua jadi 1
  prima[0]=0;
  prima[1]=0;
  for (int i=2;i*i<=n;i++) {
      if (prima[i]) {
        for (int j=i*i;j<=n;j+=i) {
          prima[j]=0;
        }
      }
  }
}
```

9 Modified Sieve

```
faktorisasi prima
int count[10005];
while (n!=1) {
       count[prima[n]]++;
       n/=prima[n];
}

number of prime divisors
for (int i=1;i<=n;i++) {
       if (prima[i]==0) {
              for (int j=i;j<=n;j+=i) {
                     prima[j]++;
              }
       }
}

number of factors
for (int i=1;i<=n;i++) {
              for (int j=i;j<=n;j+=i) {
                     prima[j]=i;
              }
}
```

10 Fast Expo

```
LL poww(LL a,LL b) {
  if (b==0) return 1;
  if (b==1) return a;
  if (b%2==0) return ((poww(a,b/2)%MOD)*(poww(a,b/2)%MOD))%
      ↪ MOD;
  else return ((((poww(a,b/2)%MOD)*(poww(a,b/2)%MOD))%MOD)*
      ↪ a)%MOD;
}
```

11 0-1 BFS

```
int n,m;
int x[4]={1,0,-1,0};
int y[4]={0,-1,0,1};
int vis[1005][1005];
char a[1005][1005];

bool inside(int a,int b) {
  if (a>=1&&a<=n&&b>=1&&b<=m) return 1;
  else return 0;
}

int main () {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n >> m;
    for (int i=1;i<=n;i++) {
      string s;
      cin >> s;
      for (int j=1;j<=m;j++) {
        a[i][j]=s[j-1];
        vis[i][j]=INT_MAX;
      }
    }
    deque<pii> q;
    q.push_front(mp(1,1));
    vis[1][1]=0;
    while (!q.empty()) {
      pii cur=q.front();
      q.pop_front();
      for (int p=0;p<4;p++) {
        int nx=cur.fi+x[p],ny=cur.se+y[p];
        if (inside(nx,ny)&&vis[nx][ny]>vis[cur.fi][cur.se])
            ↪ {
          if (a[nx][ny]==a[cur.fi][cur.se]) { //cost=0
            vis[nx][ny]=vis[cur.fi][cur.se];
            q.push_front(mp(nx,ny));
          } else if (a[nx][ny]!=a[cur.fi][cur.se]){ //cost=1
            vis[nx][ny]=vis[cur.fi][cur.se]+1;
            q.push_back(mp(nx,ny));
          }
        }
      }
    }

    cout << vis[n][m] << endl;
    return 0;
}
```

12 LCS

```
int n,m,dp[5005][5005];
string a,b;

int LCS(int p,int q) {
  if (p<=0||q<=0) return 0;
  if (dp[p][q]!=-1) return dp[p][q];
  if (a[p-1]==b[q-1]) return dp[p][q]=LCS(p-1,q-1)+1;
  else return dp[p][q]=max(LCS(p-1,q),LCS(p,q-1));
}

int main () {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n >> m;
    cin >> a >> b;
    memset (dp,-1,sizeof(dp));
    int ans=LCS(n,m);
    string answer="";
    while (n>0&&m>0) {
      if (a[n-1]==b[m-1]) {
        answer+=a[n-1];
        n--;
        m--;
      } else if (dp[n-1][m]<dp[n][m-1]) m--;
      else n--;
    }
    reverse(answer.begin(),answer.end());
    if (answer.size()>0) cout << answer << endl; else cout
        ↪ << "NONE" << endl;
    return 0;
}
```

13 LPS

```
LPS = LCS antara string dan reverse stringnya.
```

14 Random

```
srand(time(NULL)); //HARUS ADA
// generate random numbers between [a,b)
rand() % (b - a) + a;
// generate random numbers between [0,b)
rand() % b;
// generate random permutations
random_permutation(anArray, anArray + 10);
random_permutation(aVector, aVector + 10);
```

15 Prime number < 100

```
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
```

16 Leap Year

```
bool isLeap(int n)
{
    if (n%100==0)
        if (n%400==0) return true;
        else return false;
    if (n%4==0) return true;
    else return false;
}
```

17 Generate Combinations

```
// n>=m, choose M numbers from 1 to N.
void combination(int n, int m)
{
    if (n<m) return;
    int a[50]={0};
    int k=0;
    for (int i=1;i<=m;i++) a[i]=i;
    while (true) {
        for (int i=1;i<=m;i++)
            cout << a[i] << "␣";
        cout << endl;
        k=m;
        while ((k>0) && (n-a[k]==m-k)) k--;
        if (k==0) break;
        a[k]++;
        for (int i=k+1;i<=m;i++)
```

```
            a[i]=a[i-1]+1;
    }
}
```

18 Binomial Coefficient

```
#define MAXN 100 // largest n or m
long long bc[MAXN][MAXN]; //bc[n][r]=nCr
void binomial_coefficient(int n) {
    for (int i=0; i<=n; i++) bc[i][0] = 1;
    for (int j=0; j<=n; j++) bc[j][j] = 1;
    for (int i=1; i<=n; i++)
    for (int j=1; j<i; j++)
        bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
}
```

19 Bignum Multiplication - JAVA

```
// fast algorithm to find multiplication of two big numbers
    ↪ .
import java.math.BigInteger;
import java.util.Random;
class Karatsuba {
       private final static BigInteger ZERO = new
           ↪ BigInteger("0");
       public static BigInteger karatsuba(BigInteger x,
           ↪ BigInteger y)
       {
              int N = Math.max(x.bitLength(), y.bitLength
                  ↪ ());
              if (N <= 2000) return x.multiply(y);
              N=(N/2)+(N %2);
              BigInteger b = x.shiftRight(N);
              BigInteger a = x.subtract(b.shiftLeft(N));
              BigInteger d = y.shiftRight(N);
              BigInteger c = y.subtract(d.shiftLeft(N));
              BigInteger ac = karatsuba(a, c);
              BigInteger bd = karatsuba(b, d);
              BigInteger abcd = karatsuba(a.add(b), c.add(
                  ↪ d));
              return ac.add(abcd.subtract(ac).subtract(bd)
                  ↪ .shiftLeft(N)).add(bd.shiftLeft(2*N)
                  ↪ );
       }
       public static void main(String[] args)
       {
              long start, stop, elapsed;
              Random random = new Random();
              int N = Integer.parseInt(args[0]);
              BigInteger a = new BigInteger(N, random);
              BigInteger b = new BigInteger(N, random);
              start = System.currentTimeMillis();
              BigInteger c = karatsuba(a, b);
              stop = System.currentTimeMillis();
              System.out.println(stop - start);
              start = System.currentTimeMillis();
              BigInteger d = a.multiply(b);
              stop = System.currentTimeMillis();
              System.out.println(stop - start);
              System.out.println((c.equals(d)));
       }
}
```

20 Euler totient function

```
// the positive integers less than or equal to n that are
    ↪ relatively prime to n.
int phi (int n)
{
       int result = n;
       for (int i=2; i*i<=n; ++i)
       if(n %i==0)
       {
              while(n %i==0)
              n /= i;
              result -= result / i;
       }
       if (n > 1)
       result -= result / n;
       return result;
}
```

Application: $a^{\varphi(n)} \equiv 1(modn)$ kalau a dan m koprima

21 Longest increasing common sequence (LICS)

```
int a[100];
int b[100];
int f[100];
int n=0, m=0;
int main() {
       cin >> n;
       for (int i=1;i<=n;i++) cin >> a[i];
       cin >> m;
       for (int i=1;i<=m;i++) cin >> b[i];
       for (int i=1;i<=n;i++) {
              int k=0;
              for (int j=1;j<=m;j++) {
                     if (a[i]>b[j] && f[j]>k) k=f[j];
                     else if (a[i]==b[j] && k+1>f[j]) f[j
                         ↪ ]=k+1;
              }
       }
       int ans=0;
       for (int i=1;i<=m;i++)
              if (f[i]>ans) ans=f[i];
       cout << ans << endl;
       return 0;
}
```

22 LIS

```
int n=0;
int a[100], f[100], x[100];
int main() {
       cin >> n;
       for (int i=1;i<=n;i++) {
              cin >> a[i];
              x[i]=INT_MAX;
       }
       f[0]=0;
       int ans=0;
       for(int i=1;i<=n;i++) {
              int l=0, r=i;
              while (l+1<r) {
                     int m=(l+r)/2;
                     if (x[m]<a[i]) l=m; else r=m;
                     // change to x[m]<=a[i] for non-
                         ↪ decreasing case
              }
              f[i]=l+1;
              x[l+1]=a[i];
              if (f[i]>ans) ans=f[i];
       }
       cout << ans << endl;
       return 0;
}
```

23 Max sum rectangle

```
int a[150][150]={0};
int c[200]={0};
int maxarray(int n) {
       int b=0, sum=-100000000;
       for (int i=1;i<=n;i++) {
              if (b>0) b+=c[i];
              else b=c[i];
              if (b>sum) sum=b;
       }
       return sum;
}

int maxmatrix(int n) {
       int sum=-100000000, max=0;
       for (int i=1;i<=n;i++) {
              for (int j=1;j<=n;j++)
                     c[j]=0;
              for (int j=i;j<=n;j++) {
                     for (int k=1;k<=n;k++)
                            c[k]+=a[j][k];
                     max=maxarray(n);
                     if (max>sum) sum=max;
                     }
       }
       return sum;
}

int main() {
       int n=0;
       cin >> n;
       for (int i=1;i<=n;i++)
              for (int j=1;j<=n;j++)
                     cin >> a[i][j];
       cout << maxmatrix(n);
       return 0;
}
```

24 Floyd Warshall

```
// map[i][j]=infinity at start
void floyd() {
    for (int k=1; k<=n; k++)
        for (int i=1; i<=n; i++)
            for (int j=1; j<=n; j++)
                if (i!=j && j!=k && i!=k)
                    if (map[i][k]+map[k][j]<map[i][j])
                        map[i][j]=map[i][k]+map[k][j];
}
```

25 LCA+RMQ

A weighted tree is given. You must find the distance between two given nodes. Input The first line contains the number of nodes of the tree n (1 n 50000). The nodes are numbered from 0 to n 1. Each of the next n 1 lines contains three integers u, v, w, which correspond to an edge with weight w (0 w 1000) connecting nodes u and v. The next line contains the number of queries m (1 m 75000). In each of the next m lines there are two integers. Output For each range minimum query, output the distance between the nodes with the given numbers.

```
struct Graph {
    struct Edge {
        int to;
        int len;
    };

    const static int MAXNODE = 1 * 1e5 + 2;

    vector<int> g[MAXNODE];
    vector<Edge> edge;
    int n;

    int root = 0;

    void init(int nn, int m=0) {
        n = nn;
        for (int i = 0; i <= n; i++)
            g[i].clear();
        edge.clear();
        m *= 2;
        edge.reserve(m); // may speedup // add_e too slow
    }

    void add_e(int x, int y, int len) {
        g[x].push_back(edge.size());
        edge.push_back((Edge){y, len});

        g[y].push_back(edge.size());
        edge.push_back((Edge){x, len});
    }

    void show() {
        for (int i = 0; i <= n; i++) {
            printf("%d:", i);
            for (int ie : g[i])
                printf(" %d", edge[ie].to);
            printf("\n");
        }
        printf("\n");
    }

    //
    // --- start of LCA ---
    //
    vector<int> dis_to_root;
    vector<int> first_visit_time; // max possible number of
        ↪ visits to all nodes == 2 * number of nodes - 1
    vector<int> visit;
    int visit_counter;
    vector<vector<int>> rmq;

    int range_minimum_query(int l, int r) { // query [l, r]
        if (l > r)
            swap(l, r);

        int interval_len = r - l; // (r - l + 1) - 1

        int first_half = 1;
        while ((1 << first_half) <= interval_len)
            first_half++;
        first_half--;

        int second_half = r - (1 << first_half) + 1;
        if (first_visit_time[rmq[l][first_half]] <
            ↪ first_visit_time[rmq[second_half][
            ↪ first_half]])
            return rmq[l][first_half];
        return rmq[second_half][first_half];
    }

    int get_lca(int x, int y) {
        return range_minimum_query(first_visit_time[x],
            ↪ first_visit_time[y]);
    }

    int dist(int x, int y) {
        int lca = get_lca(x, y);
        return dis_to_root[x] + dis_to_root[y] - 2 *
            ↪ dis_to_root[lca];
    }

    void euler_tour(int cur) {
        visit[++visit_counter] = cur; // v_t[node] = time
            ↪ // needed in case don't have two child
        if (first_visit_time[cur] == 0) // if first time
            first_visit_time[cur] = visit_counter; // record
                ↪ time f_v_t[node] = time
        for (int ie : g[cur]) {
            const Edge& e = edge[ie];
```

```
            int nx = e.to;
            int len = e.len;
            if (first_visit_time[nx] == 0) {
                dis_to_root[nx] = dis_to_root[cur] + len;
                euler_tour(nx);
                visit[++visit_counter] = cur; // every two
                    ↪ child_visit_time have one
                    ↪ parent_visit_time inserted between
            }
        }
    }

    void build_lca() { // O(Nlog(N))
        int one_n = n + 1;
        int two_n = 2 * one_n;
        vector<int>(one_n, 0).swap(dis_to_root);
        vector<int>(one_n, 0).swap(first_visit_time);
        vector<int>(two_n, 0).swap(visit);

        int LOG_MAXLENGTH = log2(two_n) + 2;
        vector<vector<int>>(two_n, vector<int>(
            ↪ LOG_MAXLENGTH)).swap(rmq);

        visit_counter = 0;
        euler_tour(root);

        for (int i = 0; i < visit_counter; i++)
            rmq[i][0] = visit[i];

        for (int j = 1; j < LOG_MAXLENGTH; j++)
            for (int i = 0; i < visit_counter; i++) {
                if (i + (1 << j) > visit_counter)
                    break;
                rmq[i][j] = rmq[i][j - 1];
                if (first_visit_time[rmq[i][j - 1]] >
                    ↪ first_visit_time[rmq[i + (1 << (j -
                    ↪ 1))][j - 1]])
                    rmq[i][j] = rmq[i + (1 << (j - 1))][j
                        ↪ -1];
            }
    }
    //
    // --- end of LCA ---
    //
};

int n, m;
Graph g;

int main(int argc, char const *argv[]) {
    scanf("%d", &n);
    g.init(n, n);
    for (int i = 1; i < n; i++) {
        int x, y, d;
        scanf("%d␣%d␣%d", &x, &y, &d);
        g.add_e(x, y, d);
    }

    g.build_lca();

    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        int x, y;
        scanf("%d␣%d", &x, &y);
        printf("%d\n", g.dist(x, y));
    }
}
```

26 Segtree

1=Update (add Z to elements indexed X to Y), 2=max and min per query

```
const ll sz = 4e5+5;
ll seg[2][sz], lazy[sz], a[sz];
ll N, M, X, L, R, Y, Z;

void check(ll p, ll s, ll e) {
        if (lazy[p] != 0) {
                seg[0][p] += lazy[p];
                seg[1][p] += lazy[p];
                if (s != e) {
                        lazy[2*p] += lazy[p];
                        lazy[2*p+1] += lazy[p];
                }
                lazy[p] = 0;
        }
}

void build(ll p,ll s,ll e) {
        check(p,s,e);
        if (s == e) {
                seg[0][p] = a[s];
                seg[1][p] = a[s];
                return;
        }

        build(2*p,s,(s+e)/2);
        build(2*p+1,(s+e)/2+1,e);

        seg[0][p] = min(seg[0][2*p], seg[0][2*p+1]);
        seg[1][p] = max(seg[1][2*p], seg[1][2*p+1]);
}

void update(ll p,ll s,ll e,ll a,ll b,ll v) {
        check(p,s,e);
        if(s >= a && e <= b) {
                seg[0][p] += v;
                seg[1][p] += v;
                if (s != e) {
                        lazy[2*p] += v;
                        lazy[2*p+1] += v;
                }
                return;
        }
        if(s > b || e < a) {
                return;
        }
        update(2*p,s,(s+e)/2,a,b,v);
        update(2*p+1,(s+e)/2+1,e,a,b,v);
        seg[0][p] = min(seg[0][2*p], seg[0][2*p+1]);
        seg[1][p] = max(seg[1][2*p], seg[1][2*p+1]);
}

ll getMin(ll p, ll s, ll e, ll a, ll b) {
        check(p,s,e);

        if (s >= a && e <= b) {
                return seg[0][p];
        }

        if (s > b || e < a) {
                return INT_MAX;
        }

        return min(getMin(2*p,s,(s+e)/2,a,b),getMin(2*p+1,(
            ↪ s+e)/2+1,e,a,b));
}

ll getMax(ll p, ll s, ll e, ll a, ll b) {
        check(p,s,e);

        if (s >= a && e <= b) {
                return seg[1][p];
        }

        if (s > b || e < a) {
                return INT_MIN;
        }

        return max(getMax(2*p,s,(s+e)/2,a,b),getMax(2*p+1,(
            ↪ s+e)/2+1,e,a,b));
}

int main() {
        ios_base::sync_with_stdio(false);
```

```
        cin.tie(NULL);
        memset(seg,0,sizeof(seg));
        cin >> N ;
        for (ll i = 0; i < N; i++) cin >> a[i];
        cin >> M;
        build(1,0,N-1);
        while (M--) {
                int p;
                cin >> p >> X >> Y;
                X--,Y--;
                if (p==1) {
                        cin >> Z;
                        update(1,0,N-1,X,Y,Z);
                        continue;
                }
                cout << getMax(1,0,N-1,X,Y)-getMin(1,0,N-1,X
                    ↪ ,Y) << '\n';
        }
        return 0;
}
```

27 KMP

```
#define HHH 10003

int ne[HHH]; // next[], if par[i] not matched, jump to i =
    ↪ ne[i]
int kmp(string& par, string& ori) {
    ne[0] = -1;
    for (int p = ne[0], i = 1; i < par.length(); i++) {
        while (p >= 0 && par[p+1] != par[i])
            p = ne[p];
        if (par[p+1] == par[i])
            p++;
        ne[i] = p;
    }

    int match = 0;
    for (int p = -1, q = 0; q < ori.length(); q++) {
        while (p >= 0 && par[p+1] != ori[q])
            p = ne[p];
        if (par[p+1] == ori[q])
            p++;
        if (p + 1 == par.length()) { // match!
            p = ne[p];
            match++;
        }
    }

    return match; // return number of occurance
}

int main () {
    int n;
    cin >> n;
    string par, ori;
    while (cin >> par >> ori)
        cout << kmp(par, ori) << endl;
    return 0;
}
```

28 Kruskal MST

```
//Pseudocode:
// Initialize result
mst_weight = 0

// Create V single item sets
for each vertex v
        parent[v] = v;
        rank[v] = 0;

Sort all edges into non decreasing
order by weight w

for each (u, v) taken from the sorted list E
    do if FIND-SET(u) != FIND-SET(v)
        print edge(u, v)
        mst_weight += weight of edge(u, v)
        UNION(u, v)

//-- END OF PSEUDOCODE --
// C++ program for Kruskal's algorithm to find Minimum
// Spanning Tree of a given connected, undirected and
// weighted graph
#include<bits/stdc++.h>
using namespace std;

// Creating shortcut for an integer pair
typedef pair<int, int> iPair;

// Structure to represent a graph
struct Graph
{
    int V, E;
    vector< pair<int, iPair> > edges;

    // Constructor
    Graph(int V, int E)
    {
        this->V = V;
        this->E = E;
    }

    // Utility function to add an edge
    void addEdge(int u, int v, int w)
    {
        edges.push_back({w, {u, v}});
    }

    // Function to find MST using Kruskal's
    // MST algorithm
    int kruskalMST();
};

// To represent Disjoint Sets
struct DisjointSets
{
    int *parent, *rnk;
    int n;

    // Constructor.
    DisjointSets(int n)
    {
        // Allocate memory
        this->n = n;
        parent = new int[n+1];
        rnk = new int[n+1];

        // Initially, all vertices are in
        // different sets and have rank 0.
        for (int i = 0; i <= n; i++)
        {
            rnk[i] = 0;

            //every element is parent of itself
            parent[i] = i;
        }
    }

    // Find the parent of a node 'u'
    // Path Compression
    int find(int u)
    {
        /* Make the parent of the nodes in the path
           from u--> parent[u] point to parent[u] */
        if (u != parent[u])
            parent[u] = find(parent[u]);
        return parent[u];
    }

    // Union by rank
```

```
    void merge(int x, int y)
    {
        x = find(x), y = find(y);

        /* Make tree with smaller height
           a subtree of the other tree */
        if (rnk[x] > rnk[y])
            parent[y] = x;
        else // If rnk[x] <= rnk[y]
            parent[x] = y;

        if (rnk[x] == rnk[y])
            rnk[y]++;
    }
};

 /* Functions returns weight of the MST*/

int Graph::kruskalMST()
{
    int mst_wt = 0; // Initialize result

    // Sort edges in increasing order on basis of cost
    sort(edges.begin(), edges.end());

    // Create disjoint sets
    DisjointSets ds(V);

    // Iterate through all sorted edges
    vector< pair<int, iPair> >::iterator it;
    for (it=edges.begin(); it!=edges.end(); it++)
    {
        int u = it->second.first;
        int v = it->second.second;

        int set_u = ds.find(u);
        int set_v = ds.find(v);

        // Check if the selected edge is creating
        // a cycle or not (Cycle is created if u
        // and v belong to same set)
        if (set_u != set_v)
        {
            // Current edge will be in the MST
            // so print it
            cout << u << "␣-␣" << v << endl;

            // Update MST weight
            mst_wt += it->first;

            // Merge two sets
            ds.merge(set_u, set_v);
        }
    }

    return mst_wt;
}

// Driver program to test above functions
int main()
{
    /* Let us create above shown weighted
       and unidrected graph */
    int V,E;
    cin >> V >> E;
    Graph g(V, E);

    // making above shown graph
    for (int i=1;i<=E;i++) {
        int u,v,w;
        cin >> u >> v >> w;
        g.addEdge(u,v,w);
    }
    cout << "Edges␣of␣MST␣are␣\n";
    int mst_wt = g.kruskalMST();

    cout << "\nWeight␣of␣MST␣is␣" << mst_wt;

    return 0;
}
```

29 NIM Game

```
Rules of the Game of Nim: There are n piles of coins. When
    ↪ it is a players turn he chooses one pile and takes
    ↪ at least one coin from it. If someone is unable to
    ↪ move he loses (so the one who removes the last
    ↪ coin is the winner).

Let n1, n2, ..., nk, be the sizes of the piles. It is a
    ↪ losing position for the player whose turn it is if
    ↪ and only if n1 xor n2 xor ... xor nk = 0.
```

30 Convex Hull - Graham Scan

```
struct Point {
    long x;
    long y;

    bool at_right_of(Point& that, Point& base) {
        Point vec_self = {this->x - base.x, this->y - base.
            ↪ y};
        Point vec_that = {that.x - base.x, that.y - base.y
            ↪ };

        long product = vec_self * vec_that;
        if (product > 0)
            return true; // "this" is at right of "that"
        if (product == 0 && vec_self.length() > vec_that.
            ↪ length())
            return true; // "this" is at right of "that"
        return false; // "this" is NOT at right of "that"
    };
    long operator* (Point& that) {
        return this->x * that.y - this->y * that.x;
    };
    double distance_to(Point& that) {
        long x_diff = this->x - that.x;
        long y_diff = this->y - that.y;
        return sqrt(x_diff * x_diff + y_diff * y_diff);
    };
    double length() {
        return sqrt(this->x * this->x + this->y * this->y);
    }
};

Point p[1005];
int my_stack[1005];
int n, l, my_stack_top = -1;

bool compare(Point p1, Point p2) {
    return p1.at_right_of(p2, p[0]);
}

void push(int index) {
    my_stack[++my_stack_top] = index;
}
int pop() {
    int temp = my_stack[my_stack_top--];
    return temp;
}

void graham_scan() {
    push(0);
    push(1);

    int pre;
    int prepre;
    for (int i = 2; i < n; i++) {
        pre = my_stack_top;
        prepre = my_stack_top - 1;
```

```
        while (p[i].at_right_of(p[my_stack[pre]], p[
            ↪ my_stack[prepre]])) {
            pop();
            if (my_stack_top == 0)
                break;
            pre = my_stack_top;
            prepre = my_stack_top - 1;
        }
        push(i);
    }

    int last = my_stack_top;
    if (p[0].at_right_of(p[my_stack[last]], p[my_stack[pre
        ↪ ]]))
        pop();
}

int main(int argc, char const *argv[]) {
    cin >> n >> l;

    int minimun = 0;
    for (int i = 0; i < n; ++i) {
        int temp_x, temp_y;
        cin >> temp_x >> temp_y;
        p[i] = {temp_x, temp_y};

        if ((p[i].y < p[minimun].y) || (p[i].y == p[minimun
            ↪ ].y && p[i].x < p[minimun].x))
            minimun = i;
    }

    Point temp = {p[minimun].x, p[minimun].y}; // swap
        ↪ lowest and most left point to p[0]
    p[minimun] = p[0];
    p[0] = temp;

    sort(p + 1, p + n, compare); // use p[0] as base, sort
        ↪ according to polar angle
    graham_scan();
    // now all points in the stack is on Convex Hull //
        ↪ size of stack = 1 + stack_top

    for (int i = 0; i <= my_stack_top; i++)
        cout << "point␣" << my_stack[i] << "␣is␣on␣Convex␣
            ↪ Hull" << endl;
}
```

31 LOG

```
Built-in log(double) is not accurate for integer.
Should (int)(log(double)+0.000....001)
```

32 Square Root

```
long long sq(long long a) {
    long long l = 1;
    long long r = a + 1;
    while (l + 1 < r) {
        long long m = (l + r) / 2;
        if (a / m < m)
            r = m;
        else
            l = m;
    }
    return l;
}
```

33 Weighted Activity Selection

```
// C++ program for weighted job scheduling using Dynamic
// Programming and Binary Search
#include <iostream>
#include <algorithm>
using namespace std;

// A job has start time, finish time and profit.
struct Job
{
    int start, finish, profit;
};

// A utility function that is used for sorting events
// according to finish time
bool myfunction(Job s1, Job s2)
{
    return (s1.finish < s2.finish);
}

// A Binary Search based function to find the latest job
// (before current job) that doesn't conflict with current
// job. "index" is index of the current job. This function
// returns -1 if all jobs before index conflict with it.
// The array jobs[] is sorted in increasing order of finish
// time.
int binarySearch(Job jobs[], int index)
{
    // Initialize 'lo' and 'hi' for Binary Search
    int lo = 0, hi = index - 1;

    // Perform binary Search iteratively
    while (lo <= hi)
    {
        int mid = (lo + hi) / 2;
        if (jobs[mid].finish < jobs[index].start)
        {
            if (jobs[mid + 1].finish < jobs[index].start)
                lo = mid + 1;
            else
                return mid;
        }
        else
            hi = mid - 1;
    }

    return -1;
}

// The main function that returns the maximum possible
// profit from given array of jobs
int findMaxProfit(Job arr[], int n)
{
    // Sort jobs according to finish time
    sort(arr, arr+n, myfunction);

    // Create an array to store solutions of subproblems.
        ↪ table[i]
    // stores the profit for jobs till arr[i] (including
        ↪ arr[i])
    int *table = new int[n];
    table[0] = arr[0].profit;

    // Fill entries in table[] using recursive property
    for (int i=1; i<n; i++)
    {
        // Find profit including the current job
        int inclProf = arr[i].profit;
        int l = binarySearch(arr, i);
        if (l != -1)
            inclProf += table[l];

        // Store maximum of including and excluding
        table[i] = max(inclProf, table[i-1]);
    }

    // Store result and free dynamic memory allocated for
        ↪ table[]
    int result = table[n-1];
    delete[] table;
```

```
    return result;
}

// Driver program
int main()
{
    Job arr[100005];
    int n;
    cin >> n;
    for (int i=0;i<n;i++) cin >> arr[i].start >> arr[i].
        ↪ finish >> arr[i].profit;
    cout << findMaxProfit(arr, n);
    return 0;
}
```

34 Template

```
#include <bits/stdc++.h>
#define fi first
#define se second
#define pb push_back
#define mp make_pair
#define MOD 1000000007
#define pii pair<int,int>
#define LL long long

using namespace std;

int main () {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);

    return 0;
}
```