Competitive Programming - ICPC (2023)

Contents

1 Templates

1.1 C++ template

```cpp
#include <bits/stdc++.h>
using namespace std;

#define fi first
#define se second
#define pb push_back
#define endl '\n'
#define ALL(v) v.begin(), v.end()
#define SZ(arr) ((int) arr.size())

typedef long long ll;
typedef pair<int,int> ii;
typedef pair<ll,ll> pll;
typedef vector<int> vi;
typedef vector<ll> vl;

const int INF = 1e9/2-1; // ll 1e18/2-1;
const ll PI = acos(-1);
const ll EPS= 1e-9;
const int MOD = 1e9+7;
const string ABC = "abcdefghijklmnopqrstuvwxyz";

int dirx[8] = { -1, 0, 0, 1, -1, -1, 1, 1 };
int diry[8] = { 0, 1, -1, 0, -1, 1, -1, 1 };
char dirT[4] = { 'L', 'D', 'U', 'R'};

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL); cout.tie(NULL);
  cout << setprecision(20) << fixed;

  // Read and write in a file
  //freopen("problem.in", "r", stdin);
  //freopen("problem.out", "w", stdout);

  return 0;
}
```

1.2 Python template

```python
import sys
import math
import bisect
import io,os
```

```python
from sys import stdin,stdout
from math import gcd,floor,sqrt,log
from collections import defaultdict as dd
from bisect import bisect_left as bl,bisect_right as br

# profundidad maxima de la pila.
# default = 10^3
sys.setrecursionlimit(100000000)

flush  =lambda: stdout.flush()
input  =lambda: stdin.readline()
print  =lambda x: stdout.write(str(x) + "\n")
ceil   =lambda x: int(x) if(x==int(x)) else int(x)+1
seq    =lambda typ=int: list(map(typ,input().strip().
    split()))
jn     =lambda x,l: x.join(map(str,l))

mod=1000000007

# For too large inputs
input = io.BytesIO(os.read(0,os.fstat(0).st_size)).
    readline
s = input().decode() # for strings

# Notes
# Para imprimir un arreglo
print(*list)
print(jn(" ", list))
```


2 Graph algorithms

2.1 Bellman Ford

```cpp
#include "template.h";

// find shortest distance from a node x to all other
    nodes
// O(E*V) list adjacency, O(V^3) matrix
// trick: si no se modifica ninguna distancia durante una
     ronda, detenga el algoritmo

// Detectar ciclos negativos:
// Ejecutar el algoritmo n veces, si en la ultima vez se
    modifica alguna distancia
// entonces **hay** un ciclo negativo
struct EDGE { int a, b, w; };
vector<int> dist;
vector<EDGE> edges;

void bellmanFord(int x, int n) {
  dist.resize(n+1, INF);
  dist[x] = 0;

  for(int i=1; i<=n-1; ++i) {
    for(auto e: edges) {
      int a = e.a, b = e.b, w = e.w;
      dist[b] = min(dist[b], dist[a]+w); // cuidado con
          overflow !!!!
    }
  }

  bool negativeCycle = false;
  for(auto e: edges) {
    int a = e.a, b = e.b, w = e.w;
    if(dist[b] > dist[a]+w) negativeCycle = true;
  }
}
```

2.2 Dijkstra (Dense)

```cpp
#include "template.h";

// Use for dense graph --> m = n^2
vector<int> dist, p;
vector<vector<pair<int, int>>> adj;
void dijkstra(int root) { // O(n^2 + m)
  int n = 0; // adj
  dist.assign(n, INF);
  p.assign(n, -1);
  vector<bool> u(n, false);

  dist[root] = 0;
  for(int i=0; i<n; ++i) {
    int v = -1;
    for(int j=0; j<n; ++j) {
      if(!u[j] && (v == -1 || dist[j] < dist[v])) {
        v = j;
      }
    }

    if(dist[v] == INF) break;

    u[v] = true;
    for(auto e: adj[v]) {
      int to = e.first;
      int len  = e.second;

      if(dist[to] > dist[v] + len) {
        dist[to] = dist[v] + len;
        p[to] = v;
      }
    }
  }
}

vector<int> get_path(int from, int to, vector<int> &p) {
  vector<int> path;
  for(int v=to; v!=from; v=p[v]) {
    path.push_back(v);
  }
  path.push_back(from);
```

```
  reverse(ALL(path));
  return path;
}
```

2.3 Dijkstra (Sparse)

```
#include "template.h";

vector<ll> dist;
vi p;
vector<vector<ii>> adj;
void dijkstra(int root) { //O((n + m) log m)
  int n = SZ(adj);
  p.assign(n, -1);
  dist.assign(n, INF);
  priority_queue<pll, vector<pll>, greater<pll>> q;
  //priority_queue<EDGE, vector<EDGE>, decltype(&comp)> q
      (comp);

  dist[root] = 0;
  q.push({ 0, root });

  ll dv; int v;
  while(!q.empty()) {
    tie(dv, v) = q.top(); q.pop();
    if(dv != dist[v]) continue;
    for(auto u: adj[v]) {
      int to = u.first, w = u.second;
      if(dist[to] > dist[v] + w) {
        dist[to] = dist[v] + w;
        p[to] = v;
        q.push({ dist[to], to });
      }
    }
  }
}

vector<int> find_path(int from, int to, vector<int> &p) {
  vector<int> path;
  for(int v=to; v!=from; v=p[v]) path.pb(v);
  path.pb(from);
  reverse(ALL(path));
  return path;
}
```

2.4 Floyd Warshall

```
#include "template.h";
// find all shortest path
// O(n^3)
// Trick: orden correcto es KIJ, sino esta seguro,
    ejecute floydwarshall 3 veces
// Detectar ciclo negativo:
// Si al final del algoritmo la distance de un nodo x a
    si mismo
// es negativa, entonces **hay** ciclo negativo
vector<vector<int>> dist, adj, parent;
void init(int n) {
  dist = vector<vector<int>>(n+1, vector<int>(n+1, 0));
  parent = vector<vector<int>>(n+1, vector<int>(n+1, 0));

  for(int i=1; i<=n; ++i) {
    for(int j=1; j<=n; ++j) {
      parent[i][j] = i;
    }
  }

  for(int i=1; i<=n; ++i) {
    for(int j=1; j<=n; ++j) {
      if(i == j) dist[i][j] = 0;
      else if(adj[i][j]) dist[i][j] = adj[i][j];
      else dist[i][j] = INF;
    }
  }
}

void floydwarshall(int n) {
  for(int k=1; k<=n; ++k) {
    for(int i=1; i<=n; ++i) {
      for(int j=1; j<=n; ++j) {
        if (dist[i][k] < INF && dist[k][j] < INF) {
          dist[i][j] = min(dist[i][j], dist[i][k] + dist[
              k][j]);
          parent[i][j] = parent[k][j];
        }
      }
    }
  }
}

void printPath(int i, int j) {
  if(i != j) printPath(i, parent[i][j]);
}
```

2.5 Topo Sort

```
#include "template.h";
// regresa el orden topologico lexicrograficamente menor

vector<int> indegree;

void fillIndegree(vector<pair<int, int>> &edges, int n) {
  indegree.resize(n, 0);
  for(int i=0; i<edges.size(); ++i) {
    indegree[edges[i].second]++;
  }
}
```

```cpp
pair<vector<int>, bool> toposortKahn(int n, vector<vector
    <int>> &arr) {
  priority_queue<int, vector<int>, greater<int>> q;
  // queue<int> q; --> no asegura el lexicograficamente
     menor
  vector<int> topo;

  for(int i=0; i<n; ++i) {
    if(indegree[i] == 0) {
      q.push(i); // se puede procesar
    }
  }

  while(!q.empty()) {
    int curr = q.top(); q.pop();
    topo.push_back(curr);

    for(int adj: arr[curr]) {
      indegree[adj]--; // virutally remove curr --> adj
      if(indegree[adj] == 0) q.push(adj);
    }
  }

  bool thereAreCycle = false;
  if(topo.size() != n) thereAreCycle = true;
  return {topo, thereAreCycle};
}

//in main --> fillIndegree(edges, n);
// tambien se puede calcular el indegree con dfs
```

3 Data Structures

3.1 Disjoint Set Union

```cpp
#include "template.h";

// Trick
// - Cada nodo guarda -1 como flag inicial
// Cada hijo va a guardar el indice de su padre
// Cada padre-maximo (root del grupo) va a tener el
    tamano del grupo. pero en negativo
//  asi, cuando se encuentre un valor negativo es porque
    estamos frente a un root, y al multiplicarlo por
//  -1 obtenemos el tamano del grupo

struct DSU {
        vector<int> e;
        DSU(int N) { e = vector<int>(N, -1); }

        //* get representive component (uses path
            compression)
        int find(int x) { return e[x] < 0 ? x : e[x] =
            find(e[x]); }

        bool isSameSet(int a, int b) { return find(a) ==
            find(b); }

        int getSize(int x) { return -e[find(x)]; }

        bool merge(int x, int y) {
                x = find(x), y = find(y);
                if (x == y) return false;
                if (e[x] > e[y]) swap(x, y); //* Union by
                    size (by rank)
                e[x] += e[y];
                e[y] = x;
                return true;
        }
};
```

3.2 Segment Tree

```cpp
#include "template.h";

const int N = 1e5;  // limit for array size
int n;  // array size
int t[2 * N];

int op(int a, int b) { return a + b; };

void build() {  // build the tree
  for (int i = n - 1; i > 0; --i) t[i] = op(t[i<<1], t[i
      <<1|1]);
}

void modify(int p, int value) {  // set value at position
    p
  for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] = op(t[
      p], t[p^1]);
}

int query(int l, int r) {  // sum on interval [l, r)
  int res = 0; // init INF para max/min
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) res = op(res, t[l++]);
    if (r&1) res = op(res, t[--r]);
  }
  return res;
}

int main() {
  cin >> n;
  for (int i = 0; i < n; ++i) cin >> t[n+i];
  build();

  // modify(0, 1);
  // cout << query(3, 11) << endl;
  return 0;
}
```

3.3 Segment Tree Lazy

```
#include "template.h";

// Segment tree with lazy propagation
struct segtree{
  segtree *left;
  segtree *right;
  int l, r;
  ll value, lazy;
  ll nullValue = -INF; // -inf for max, inf for min, etc

  segtree(vl &v, int l, int r) : l(l), r(r) {
    int m = (l+r)>>1;
    lazy = 0ll;
    if (l!=r) {
      left = new segtree(v, l, m);
      right = new segtree(v, m+1, r);
    } else value = v[l];
  }

  ll operation(ll leftValue, ll rightValue) { return
     leftValue+rightValue; } // change operation!!!!

  void propagate() {
    if (lazy) {
      value += lazy*(r-l+1); // += or = ???
      if (l!=r) left->lazy+=lazy, right->lazy+=lazy;
      lazy = 0;
    }
  }

  ll get(int a, int b) {
    propagate();
    if (l>b || r<a) return nullValue;
    if (l>=a && b>=r) return value;
    return operation(left->get(a,b), right->get(a,b));
  }

  void update(int a, int b, ll nv) { // nv -> new value
    propagate();
    if (l>b || r<a) return;
    if (l>=a && b>=r) {
      // value = nv;
      lazy += nv;
      propagate();
      return;
    }
    left->update(a,b,nv);
    right->update(a,b,nv);
    value = operation(left->value, right->value);
  }
};
```

4 Strings

4.1 KMP

```
#include "template.h";

vi get_phi(string &s) { // O(|s|)
  int n = SZ(s);
  vi phi(n, 0); // proper suffix - preffix [0 ... i]
  for(int i=1, j=0; i<n; ++i) {
      while(j > 0 && s[i] != s[j]) j = phi[j-1];
      if(s[i] == s[j]) j++;
      phi[i] = j;
  }
  return phi;
}

void kmp(string &t, string &p){ // O(|t| + |p|)
  vi phi = get_phi(p);
  int matches = 0;
  for(int i = 0, j = 0; i < SZ(t); ++i ) {
    while(j > 0 && t[i] != p[j] ) j = phi[j-1];
    if(t[i] == p[j]) ++j;
    if(j == SZ(p)) {
      matches++;
      j = phi[j-1];
    }
  }
}
```

4.2 Z algorithm

```
#include "template.h";

vi z_algo(string &s) { // O(|S|)
  int n = SZ(s), l=0, r=0;
  vi z(n, 0);
  for(int i=1; i<n; ++i) {
    if(i < r) z[i] = min(r-i, z[i-l]);
    while(i+z[i] < n && s[z[i]] == s[i+z[i]]) z[i]++;
    if(z[i]+i > r) { l = i; r = z[i]+i; }
  }
  return z;
}
```

5 Math

5.1 Binary Exponentiation

```
// O(log n)
```

```
// if m if prime, use a^(n mod (m-1)) instead a^n
long long binpow(long long a, long long b, long long m  =
    1) {
  a %= m;
  long long res = 1;
  while (b > 0) {
    if (b & 1) res = res * a % m;
    a = a * a % m;
    b >>= 1;
  }
  return res;
}
```

5.2 Combinatoria

```
// Opcion 1
const int MX = 1817;
long long MOD = 1000000000+7;
int C[MX + 1][MX + 1];
void nCk() {
  C[0][0] = 1;
  aux[1] = 1;
  for (int n = 1; n <= MX; ++n) {
    C[n][0] = C[n][n] = 1;
      for (int k = 1; k < n; ++k) {
        C[n][k] = (C[n - 1][k - 1] % MOD + C[n - 1][k] %
            MOD ) % MOD;
      }
  }
}

// Opcion 3
ll fact[nax];
ll binpow(ll b, ll e) {
        ll ans = 1;
        for (; e; b = b * b % MOD, e /= 2)
                if (e & 1LL) ans = ans * b % MOD;
        return ans;
}

ll C(int n, int k){
  if(n<k) return 0;
  return (fact[n] * binpow(fact[n-k], MOD-2) %MOD) *
     binpow(fact[k], MOD-2)%MOD;
}
fact[0] = 1;
    for(int i=1; i<nax; ++i) fact[i] = (1LL*i*fact[i-1])%
       MOD;
```

5.3 Count primes

```
// Same complexity of the normal sieve
// but in memory is O(sqrt(n) + S)
int count_primes(int n) {
  const int S = 10000;

  vector<int> primes;
  int nsqrt = sqrt(n);
  vector<char> is_prime(nsqrt + 2, true);
  for (int i = 2; i <= nsqrt; i++) {
    if (is_prime[i]) {
      primes.push_back(i);
      for (int j = i * i; j <= nsqrt; j += i)
        is_prime[j] = false;
    }
  }

  int result = 0;
  vector<char> block(S);
  for (int k = 0; k * S <= n; k++) {
    fill(block.begin(), block.end(), true);
    int start = k * S;
    for (int p : primes) {
      int start_idx = (start + p - 1) / p;
      int j = max(start_idx, p) * p - start;
      for (; j < S; j += p)
        block[j] = false;
    }
    if (k == 0)
      block[0] = block[1] = false;
    for (int i = 0; i < S && start + i <= n; i++) {
      if (block[i])
        result++;
    }
  }
  return result;
}
```

5.4 Factorial mod p

```
// O(logp(n))
int factmod(int n, int p) {
  vector<int> f(p);
  f[0] = 1;
  for (int i = 1; i < p; i++)
    f[i] = f[i-1] * i % p;

  int res = 1;
  while (n > 1) {
    if ((n/p) % 2)
      res = p - res;
    res = res * f[n%p] % p;
    n /= p;
  }
  return res;
}
```

5.5 Integer Factorization

```
// O(sqrt(n) / 2)
vector<long long> trial_division2(long long n) {
  vector<long long> factorization;
  while (n % 2 == 0) {
    factorization.push_back(2);
    n /= 2;
  }
  for (long long d = 3; d * d <= n; d += 2) {
    while (n % d == 0) {
      factorization.push_back(d);
      n /= d;
    }
  }
  if (n > 1) factorization.push_back(n);
  return factorization;
}

// Precompute primes O(sqrt(n) using linear sieve)
vector<long long> primes; // to sqrt(n)
vector<long long> trial_division4(long long n) {
    vector<long long> factorization;
    for (long long d : primes) {
      if (d * d > n) break;
      while (n % d == 0) {
        factorization.push_back(d);
        n /= d;
      }
    }
    if (n > 1) factorization.push_back(n);
    return factorization;
}
```

5.6 Linear Sieve of Eratosthenes

```
// O(n)
const int N = 10000000;
vector<int> lp(N+1, 0);
vector<int> pr;

for (int i=2; i <= N; ++i) {
  if (lp[i] == 0) {
    lp[i] = i;
    pr.push_back(i);
  }
  for (int j=0; j < (int)pr.size() && pr[j] <= lp[i] && i
     *pr[j] <= N; ++j) {
    lp[i * pr[j]] = pr[j];
  }
}
```

5.7 Modular division

```
int modInverse(int b, int m)
{
    int x, y; // used in extended GCD algorithm
    int g = gcdExtended(b, m, &x, &y);

    // Return -1 if b and m are not co-prime
    if (g != 1)
        return -1;

    // m is added to handle negative x
    return (x%m + m) % m;
}

// Function to compute a/b under modulo m
int modDivide(int a, int b, int m)
{
    a = a % m;
    int inv = modInverse(b, m);
    if (inv == -1)
      return -1;
    else
       return (inv * a) % m;
}

// C function for extended Euclidean Algorithm (used to
// find modular inverse.
int gcdExtended(int a, int b, int *x, int *y) {
    if (a == 0)
    {
        *x = 0, *y = 1;
        return b;
    }

    int x1, y1; // To store results of recursive call
    int gcd = gcdExtended(b%a, a, &x1, &y1);

    // Update x and y using results of recursive
    // call
    *x = y1 - (b/a) * x1;
    *y = x1;

    return gcd;
}
```

5.8 Primality tests

```
// O(sqrt(n))
bool isPrime(int x) {
  for (int d = 2; d * d <= x; d++) {
    if (x % d == 0) return false;
  }
  return true;
}
```

```
// O(iter * log2(n))
bool probablyPrimeFermat(int n, int iter=5) {
  if (n < 4) return n == 2 || n == 3;

  for (int i = 0; i < iter; i++) {
    int a = 2 + rand() % (n - 3);
    if (binpow(a, n - 1, n) != 1)
      return false;
  }
  return true;
}
```

5.9 Segmented Sieve

```
// O((R-L+1)*log(log(R)) + sqrt(R)*log(log(R)))
vector<char> segmentedSieve(long long L, long long R) {
  // generate all primes up to sqrt(R)
  long long lim = sqrt(R);
  vector<char> mark(lim + 1, false);
  vector<long long> primes;
  for (long long i = 2; i <= lim; ++i) {
    if (!mark[i]) {
      primes.emplace_back(i);
      for (long long j = i * i; j <= lim; j += i)
        mark[j] = true;
    }
  }

  vector<char> isPrime(R - L + 1, true);
  for (long long i : primes)
    for (long long j = max(i * i, (L + i - 1) / i * i); j
        <= R; j += i)
      isPrime[j - L] = false;
  if (L == 1)
    isPrime[0] = false;
  return isPrime;
}
```

5.10 Sieve of Eratosthenes

```
// O(n log log n)
// this implementation is O(n log log sqrt(n))
int n;
vector<bool> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;

for (int i = 2; i * i <= n; i++) {
  if (is_prime[i]) {
    for (int j = i * i; j <= n; j += i) {
      is_prime[j] = false;
    }
  }
}
```

6 Dynamic Programming

6.1 Binomial coefficient

```
#include "template.h";

// Using pascal triangle
const int MX = 1000;
long long MD = 999999937;
int C[MX + 1][MX + 1];

void nCk() {
  C[0][0] = 1;
  for(int n=1; n <= MX; ++n) {
    C[n][0] = C[n][n] = 1;
    for(int k=1; k<n; ++k) {
      C[n][k] = (C[n-1][k-1] % MOD + C[n-1][k] % MOD) %
          MOD;
    }
  }
}
```

6.2 LIS (nlogn)

```
#include "template.h";

// O(n * log n)
int lis(vector<int> const& a) {
  int n = a.size();
  const int INF = 1e9;
  vector<int> d(n+1, INF);
  d[0] = -INF;

  for (int i = 0; i < n; i++) {
    int j = upper_bound(d.begin(), d.end(), a[i]) - d.
        begin();
    if (d[j-1] < a[i] && a[i] < d[j]) d[j] = a[i];
  }

  int ans = 0;
  for (int i = 0; i <= n; i++) {
    if (d[i] < INF) ans = i;
  }
  return ans;
}
```

6.3 LIS

```cpp
#include "template.h";

// O(n^2)
int lis(vector<int> const& a) {
  int n = a.size();
  vector<int> d(n, 1);
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
      if (a[j] < a[i])
        d[i] = max(d[i], d[j] + 1);
    }
  }

  int ans = d[0];
  for (int i = 1; i < n; i++) {
    ans = max(ans, d[i]);
  }
  return ans;
}

// restoring path
vector<int> lis(vector<int> const& a) {
  int n = a.size();
  vector<int> d(n, 1), p(n, -1);
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
      if (a[j] < a[i] && d[i] < d[j] + 1) {
        d[i] = d[j] + 1;
        p[i] = j;
      }
    }
  }

  int ans = d[0], pos = 0;
  for (int i = 1; i < n; i++) {
    if (d[i] > ans) {
      ans = d[i];
      pos = i;
    }
  }

  vector<int> subseq;
  while (pos != -1) {
    subseq.push_back(a[pos]);
    pos = p[pos];
  }
  reverse(subseq.begin(), subseq.end());
  return subseq;
}
```

6.4 Max submatrix 2d

```cpp
#include "template.h";

//O(n^2)
// max matriz zeros, ponga todos los demas en negativo y
   encuentre el de suma maxima
// que debe ser 0
int maxSubRect = -127*100*100; // the lowest possible
   value for this problem
for(int i = 0; i < n; i++) for (int j = 0; j < n; j++) {
   // start coordinate
  for (int k = i; k < n; k++) for (int l = j; l < n; l++)
     { // end coord
    subRect = A[k][l]; // sum of all items from (0, 0) to
         (k, l): O(1)
    if (i > 0) subRect -= A[i - 1][l]; // O(1)
    if (j > 0) subRect -= A[k][j - 1]; // O(1)
    if (i > 0 && j > 0) subRect += A[i - 1][j - 1]; // O
       (1)
    maxSubRect = max(maxSubRect, subRect); // the answer
       is here
  }
}
```

7 Miscellaneous

7.1 C++ bitwise

```cpp
mask |= (1<<n) // PRENDER BIT-N
mask ^= (1<<n) // FLIPPEAR BIT-N
mask &= ~(1<<n) // APAGAR BIT-N
if(mask&(1<<n)) // CHECKEAR BIT-N
```

7.2 Subsets

```cpp
// O(n * 2^n)
void subsets(int S[], int n) {
  int nSub = 1<<n;
  for(int i=0; i<nSub; i++){
    cout<<"{ ";
    for(int k=0; k<n; k++){
      //if the k-th bit is set
      if( (1<<k) & i){
        cout<<S[k]<<" ";
      }
    }
    cout<<"}\n";
  }
}
```

7.3 C++ tricks

```cpp
// log base 2 of 32 bit integer
int log2(int x) {
```

```
  int res = 0;
  while (x >>= 1) res++;
  return res;
}

// is power of 2
bool isPowerof2(int x) {
  return (x && !(x & x-1));
}

// Most significative digit
int MSD(int n) {
  if(n == 0) return 0;

  int k = log10(n);
  int x = pow(10,k);
  int ans = n/x;

  return ans;
}

// Number of digits
int NOF (int n) {
  return floor(log10(n)) + 1;
}

// Perfect square
bool isPerfectSquare(int x) {
  int s = sqrt(x);
  return (s * s == x);
}

// Use C++11 inbuilt algorithms
all_of(first, first+n, ispositive());
any_of(first, first+n, ispositive());
none_of(first, first+n, ispositive());

// Fast multiplication or division by 2
n = n << 1 // Mutiply
n = n >> 1 // Divide
```



```
// Custom comparator sort

// Notes:
// 1. Use emplace_back() instead push_back()
```

7.4 Python tricks

```
import sys
import itertools

# Checking Memory Usage of Any Object
dic = {'a': 3, 'b': 2, 'c': 1, 'd': 1}
print(sys.getsizeof(dic))

# Sort List of Tuples by Any Index of Tuple Value
val = [('first', 3, 9), ('second', 4, 6), ('third', 2, 3)
    ]
val.sort(key = lambda x: x[2], reverse=False)
print(val)

# Wise Use of Python Dictionary Comprehension
dic = [(str(i), i*2) for i in range(5)]
print(dict(dic))

# Permutation (list_elements, #elements_x_group)
list(itertools.permutations('HAPPY', 2))

# combitation with repetitions (list_elements, #
    elements_x_group)
list(itertools.combinations_with_replacement('HAPPY', 2))

# combitation without repetitions (list_elements, #
    elements_x_group)
list(itertools.combinations('HAPPY', 2))
```