Ex-usuários de Python — CIn - UFPE

Contents

1 Graphs

1.1 Breadth First Search

```
// Time Complexity: O(V + E)
void bfs(vector<vector<int>>& adj, vector<bool>& visited, int start) {
    queue<int> operation_order;

    visited[start] = true;
    operation_order.push(start);

    while (!operation_order.empty()) {
        auto top = operation_order.front();
        operation_order.pop();

        for (auto e : adj[top]) {
            if (!visited[e]) {
                visited[e] = true;
                operation_order.push(e);
            }
        }
    }
}
```

1.2 Depth First Search

```
// Time Complexity: O(V + E)
void dfs(vector<vector<int>>& adj, vector<bool>& visited, int v) {
    visited[v] = true;
    // pre-visited

    for (auto e: adj[v]) {
        if (!visited[e]) {
            dfs(adj, visited, e);
        }
    }

    // post-visited
}
```

1.3 TopoSort

```
// Time Complexity: O(V + E)
void toposort(vector<vector<int>>& adj, stack<int>& topo, vector<bool>&
     visited, int v) {
    visited[v] = true;
    for (auto e: adj[v]) {
        if (!visited[e]) {
            toposort(adj, topo, visited, e);
        }
    }
    topo.push(v);
}

// Time Complexity: O(V + E)
void toposort(vector<vector<int>>& adj, vector<int>& indegree, int n) {
    queue<int> q; // Use a min heap for lexicographically smallest toposort
    for (int i = 0; i < n; i++) {
        if (indegree[i] == 0) {
            q.push(i);
        }
    }

    while (!q.empty()) {
        int v = q.front();
        q.pop();
        cout << v << " ";
        for (auto e: adj[v]) {
            indegree[e]--;
            if (indegree[e] == 0) {
                q.push(e);
            }
        }
    }
}
```

1.4 Is Bicolorable

```
// Time Complexity: O(V + E)
bool bicolorable(vector<vector<int>>&adj, vector<bool>& visited, vector<
     bool>& color, int v) {
    visited[v] = true;

    for (auto e: adj[v]) {
        if (!visited[e]) {
            color[e] = !color[v];
            if (!bicolorable(adj, visited, color, e)) {
                return false;
            }
        } else if (color[e] == color[v]) {
            return false;
        }
    }
```

```
        return true;
    }
```

1.5 Dijkstra

```
// Time Complexity: O((V + E) * log(V))
void dijkstra(vector<vector<pair<int, int>>>& adj, vector<int>& dist, int s
    ) {
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int
        , int>>> pq;
    pq.push({0, s});
    dist[s] = 0;

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

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

            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}
```

1.6 Floyd Warshall

```
// Time Complexity: O(V^3)
void FloydWashall(vector<vector<int>>& dist, int n) {
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
}
```

2 Number Theory

2.1 Digit Sum

```
int digit_sum(int n) {
    while(n>=10) {
            int temp = 0;
            while (n > 0) {
                temp += n % 10;
                n /= 10;
            }
            n = temp;
        }
    return n;
}
```

2.2 Binary Search

```
// Time Complexity: O(log(n))
int binarySearch(int l, int r, int* arr, int target) {
    int answ;
    while (l <= r) {
        int m = l + (r - l) / 2;

        // If NOT SOLVE, ignore left half
        if (! (arr[m] > target) )
            l = m + 1;

        // If SOLVE, ignore right half
        else {
            answ = m;
            r = m - 1;
        }
    }

    return answ-1;
}
```

2.3 Fast Exponentiation

```
const ll MOD = 1e9+7;

class Matrix{
        public:
        vector<vector<ll>> mat;
        int m;
        Matrix(int m): m(m) {
                mat.resize(m);
                for(int i = 0; i < m; i++) mat[i].resize(m,0);
        }
        Matrix operator * (const Matrix& rhs) {
                Matrix ans = Matrix(m);
                for(int i = 0; i < m; i++)
                        for(int j = 0; j < m; j++)
                                for(int k = 0; k < m; k++)
                                        ans.mat[i][j] = (ans.mat[i][j] + (
                                            mat[i][k] * rhs.mat[k][j]) %
                                            MOD) % MOD;
                return ans;
        }
};

Matrix fexp(Matrix a, ll n) {
        int m = a.m;
        Matrix ans = Matrix(m);
        for(int i = 0; i < m; i++) ans.mat[i][i] = 1;
        while(n) {
                if(n & 1) ans = ans * a;
                a = a * a;
                n >>= 1;
        }
        return ans;
}

// Time complexity: O(log(n))
ll fexpll(ll a, ll n) {
        ll ans = 1;
        while(n) {
                if(n & 1) ans = (ans * a) % MOD;
                a = (a * a) % MOD;
                n >>= 1;
        }
        return ans;
}
```

2.4 GCD and LCM

```
// Time Complexity: O(log(min(m, n)))
ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }

// Time Complexity: O(log(min(m, n)))
ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }
```

2.5 Sieve of Eratosthenes

```
vector<ll> prime_list;

// Time Complexity: O(n log log n)
void EratosthenesSieve(ll n) {
    vector<bool> prime(n + 1, true);

    for (ll p = 2; p <= n; p++) {
        if (prime[p] == true) {
            prime_list.push_back(p);

            for (ll i = p; i <= n; i += p)
                prime[i] = false;
        }
    }
}
```

2.6 Modular Inverse

```
ll modInverse(ll n) {
    ll ex = MOD-2, result = 1;
    while (ex > 0) {
        if (ex % 2 == 1) {
            result = (result*n) % MOD;
        }
        n = (n*n) % MOD;
        ex /= 2;
    }
    return result;
}
```

3 Data Structures

3.1 Segment Tree

```
const int INF = INT_MAX;
const int max_size = 2e5 + 5;
vector<ll> seg(4 * max_size);
vector<ll> arr(max_size);

int n, q;

ll operation(ll a, ll b) { return a + b; }

// Time complexity: O(n)                    // build()
void build(int l = 0, int r = n - 1, int index = 0) {
    if (l == r) {
        seg[index] = arr[l];
        return;
    }
    int mid = l + (r - l) / 2;
    int left = 2 * index + 1;
    int right = 2 * index + 2;
    build(l, mid, left);
    build(mid + 1, r, right);
    seg[index] = operation(seg[left], seg[right]);
}

// Time complexity: O(log(n))                // query(L-1, R-1)
ll query(int L, int R, int l = 0, int r = n - 1, int index = 0) {
    if (R < l || L > r) return 0;  // Neutral element of the operation
    if (L <= l && r <= R) return seg[index];

    int mid = l + (r - l) / 2;
    int left = 2 * index + 1;
    int right = 2 * index + 2;
    ll ql = query(L, R, l, mid, left);
    ll qr = query(L, R, mid + 1, r, right);
    return operation(ql, qr);
}

// Time complexity: O(log(n))                 // update(pos-1, value)
void update(int pos, int num, int l = 0, int r = n - 1, int index = 0) {
    if (l == r) {
        seg[index] = num;
        return;
    }
    int mid = l + (r - l) / 2;
    int left = 2 * index + 1;
    int right = 2 * index + 2;
    if (pos <= mid) {
        update(pos, num, l, mid, left);
    } else
        update(pos, num, mid + 1, r, right);
    seg[index] = operation(seg[left], seg[right]);
}
```

3.2 Binary Indexed Tree (BIT)

```
const int max_size = 2e5+5;
vector<ll> arr(max_size+1,0);
vector<ll> bit(max_size+1,0);

int n, q;

// Time complexity: O(log(n))
ll query(int i) { // [1,i]
        ll ret = 0;
        for(; i > 0; i -= i & -i) {
                ret += bit[i];
        }
        return ret;
}

// Time complexity: O(log(n))
ll queryRange(int l, int r) { // [l,r]
        ll qr = query(r);
        ll ql = query(l-1);
        return qr-ql;
}

// Time complexity: O(log(n))
void increment(ll index, ll value) {
        for(; index <= n; index += index & -index) {
                bit[index] += value;
        }
}

// Time complexity: O(n * log(n))
void build(const vector<ll>& nums) {
        for(int i = 0; i < nums.size(); i++) {
                increment(i+1,nums[i]);
        }
}
```

4 Combinatorics

4.1 Factorial

```
ll fact (ll n) {
    ll answ = 1;
    for (int i = 2; i <= n; i++) {
        answ  = (answ * i) % MOD;
    }
    return answ % MOD;

}
```

4.2 Sum of PA

```
ll sumofpa(ll k, ll n) { return ((k + n) * (n - k + 1)) / 2ll; }
```

5 Dynamic Programming

5.1 Knapsack

```
    #include <bits/stdc++.h>

#define ll long long

using namespace std;

struct item {
    int weight;
      ll value;
};

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    int item_count, capacity;

    cin >> item_count >> capacity;
    ll table[++item_count][++capacity];

    int weight;
     ll  value;

    vector<item> items = {{0,0}};

    for (int i = 1; i < item_count; i++) { // reading the items
        cin >> weight >> value;

        items.push_back({weight, value});
    }

    for (int i = 0; i < item_count;i++) { // setting the capacity 0 to 0
         value;
        table[i][0] = 0;
    }
    for (int j = 0; j < capacity; j++) { // setting the item 0 to 0 value;
        table[0][j] = 0;
    }

    for (int i = 1; i < item_count; i++) { // populating the table;
        for (int j = 1; j < capacity; j++) {
            int w = items[i].weight;
             ll v = items[i].value;

            table[i][j] = table[i-1][j];
            if (w <= j) table[i][j] = max(table[i][j], v + table[i-1][j-w])
                ;
        }
    }

    vector<int> chosen_itens; // retrieving the chosen itens in the optimal
         solution
    int c = capacity - 1;
    for (int i = item_count - 1; i > 0; i--) {
        if (table[i][c] != table[i-1][c]) {
            chosen_itens.push_back(i);
            c -= items[i].weight;
        }
    }

    for (int i = 0; i < item_count; i++) { // printing the table for debug
        cout << '\n';
        for (int j = 0; j < capacity; j++) {
            cout << table[i][j] << ' ';
        }
    }
    cout << '\n';

    cout << table[--item_count][--capacity] << '\n'; //print the max value;

    for (auto e: chosen_itens) cout << e << ' '; //print the chosen itens;

    return 0;
}
```

6 Geometry

6.1 Point

```
// hypot, atan2, gcd
const double PI = acos(-1);
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<typename T>
struct PT{
  T x, y;
  PT(T x=0, T y=0) : x(x),y(y){}
  bool operator < (PT o) const { return tie(x,y) < tie(o.x,o.y); }
  bool operator == (PT o) const { return tie(x,y) == tie(o.x,o.y); }
  PT operator + (PT o) const { return PT(x+o.x,y+o.y); }
  PT operator - (PT o) const { return PT(x-o.x,y-o.y); }
  PT operator * (T k) const { return PT(x*k,y*k); }
  PT operator / (T k) const { return PT(x/k,y/k); }
  T cross(PT o) const { return x*o.y - y*o.x; }
  T cross(PT a, PT b) const { return (a-*this).cross(b-*this); }
  T dot(PT o) const { return x*o.x + y*o.y; }
  T dist2() const { return x*x + y*y; }
  double len() const { return hypot(x,y); }
  PT perp() const { return PT(-y,x); }
  PT rotate(double a) const { return PT(x*cos(a)-y*sin(a), x*sin(a)+y*cos(a
      )); }
};
ostream &operator<<(ostream &os, const PT<ll> &p) {
  return os << "(" << p.x << "," << p.y << ")";
}
```

6.2 Convex Hull

```
// retorna poligono no sentido anti horario, trocar pra < se quiser horario
```

```
template<typename T>
vector<PT<T>> convexHull(vector<PT<T>>& pts, bool sorted = false){
  if(!sorted) sort(begin(pts),end(pts));
  vector<PT<T>> h;
  h.reserve(pts.size() + 1);
  for(int it = 0; it < 2; it++){
    int start = h.size();
    for(PT<T>& c : pts){
      while((int)h.size() >= start + 2){
        PT<T> a = h[h.size()-2], b = h.back();
        // '>=' pra nao descartar pontos colineares
        if((b-a).cross(c-a) > 0) break;
        h.pop_back();
      }
      h.push_back(c);
    }
    reverse(begin(pts),end(pts));
    h.pop_back();
  }
  if(h.size() == 2 && h[0] == h[1]) h.pop_back();
  return h;
}

// nao funciona se tem pontos colineares!!!!
// considera ponto na aresta como dentro
template<typename T>
bool isInside(vector<PT<T>>& hull, PT<T> p) {
  int n = hull.size();
  PT<T> v0 = p - hull[0], v1 = hull[1] - hull[0], v2 = hull[n-1] - hull[0];
  if(v0.cross(v1) > 0 || v0.cross(v2) < 0){
    return false;
  }
  int l = 1, r = n - 1;
  while(l != r){
    int mid = (l + r + 1) / 2;
    PT<T> v0 = p - hull[0], v1 = hull[mid] - hull[0];
    if(v0.cross(v1) < 0)
      l = mid;
    else
      r = mid - 1;
  }
  v0 = hull[(l+1)%n] - hull[l], v1 = p - hull[l];
  return v0.cross(v1) >= 0;
}

// poligonos
ll polygon_area_db(const vector<Point>& poly){
  ll area = 0;
  for(int i = 0, n = (int)poly.size(); i < n; ++i) {
    int j = i + 1 == n ? 0 : i + 1;
    area += cross(poly[i], poly[j]);
  }
  return abs(area);
}
// Teorema de Pick para lattice points
// Area = insidePts + boundPts/2 - 1
// 2A - b + 2 = 2i
// usar gcd dos lados pra contar bound pts
ll cntInsidePts(ll area_db, ll bound){
  return (area_db + 2LL - bound)/2;
}
```

6.3 Closest Pair

```
pii ClosestPair(vector<PT<ll>>& pts) {
  ll dist = (pts[0]-pts[1]).dist2();
  pii ans(0, 1);
  int n = pts.size();
  vector<int> p(n);
  iota(begin(p),end(p),0);
  sort(p.begin(), p.end(), [&](int a, int b) { return pts[a].x < pts[b].x;
      });
  set<pii> points;
  auto sqr = [](long long x) -> long long { return x * x; };
  for(int l = 0, r = 0; r < n; r++) {
    while(sqr(pts[p[r]].x - pts[p[l]].x) > dist) {
      points.erase(pii(pts[p[l]].y, p[l]));
      l++;
    }
    ll delta = sqrt(dist) + 1;
    auto itl = points.lower_bound(pii(pts[p[r]].y - delta, -1));
    auto itr = points.upper_bound(pii(pts[p[r]].y + delta, n + 1));
    for(auto it = itl; it != itr; it++) {
      ll curDist = (pts[p[r]] - pts[it->second]).dist2();
      if(curDist < dist) {
        dist = curDist;
        ans = pii(p[r], it->second);
      }
    }
    points.insert(pii(pts[p[r]].y, p[r]));
  }
  if(ans.first > ans.second)
    swap(ans.first, ans.second);
  return ans;
}
```