Team Notebook

omsim

December 13, 2022

Contents

1 Data Structures

1.1 Doubly Linked List

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None
    def get_data(self):
        return self.data
class Sentinel_DLL:
    def __init__(self):
        self.sentinel = Node(None)
        self.sentinel.next = self.sentinel
        self.sentinel.prev = self.sentinel
    def first_node(self):
        if self.sentinel.next == self.sentinel:
            return None
        else:
            return self.sentinel.next
    def insert_after(self, x, data):
        y = Node(data)
        y.prev = x
        y.next = x.next
        x.next = y
        y.next.prev = y
    def append(self, data):
        last_node = self.sentinel.prev
        self.insert_after(last_node, data)
    def prepend(self, data):
        self.insert_after(self.sentinel, data)
    def delete(self, x):
        x.prev.next = x.next
        x.next.prev = x.prev
    def find(self, data):
        self.sentinel.data = data
        x = self.first_node()
        while x.data != data:
            x = x.next
        self.sentinel.data = None
        if x == self.sentinel:
            return None
        else:
            return x
    def __str__(self):
        s = "["
        x = self.sentinel.next
        while x != self.sentinel:
            if type(x.data) == str:
                s += "'"
            s += str(x.data)
            if type(x.data) == str:
                s += "'"
            if x.next != self.sentinel:
                s += ", "
            x = x.next

        s += "]"
        return s
#test
llist = Sentinel_DLL()
llist.append(5)
llist.append(6)
llist.append(2)
llist.prepend(19)
print(llist)
#insert_after = insert a new node with data after node x
#append = insert new node at end of list
#prepend = insert a new node at the start of the list
#delete = delete node x
#find = finds x (note: O(n) )
```

1.2 Segment Trees

```
N = 100000;
tree = [0] * (2 * N);
def build(arr) :
    for i in range(n) :
        tree[n + i] = arr[i];
    for i in range(n - 1, 0, -1) :
        tree[i] = tree[i << 1] + tree[i << 1 | 1];
def updateTreeNode(p, value) :
    tree[p + n] = value;
    p = p + n;
    i = p;
    while i > 1 :
        tree[i >> 1] = tree[i] + tree[i ^ 1];
        i >>= 1;
def query(l, r) :
    res = 0;
    l += n;
    r += n;
    while l < r :
        if (l & 1) :
            res += tree[l];
            l += 1
        if (r & 1) :
            r -= 1;
            res += tree[r];
        l >>= 1;
        r >>= 1
    return res;
if __name__ == "__main__" :
    a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12];
    n = len(a);
    # build tree
    build(a);
    # print the sum in range(1,2) index-based
    print(query(1, 3));
    # modify element at 2nd index
    updateTreeNode(2, 1);
    # print the sum in range(1,2) index-based
    print(query(1, 3));
```

1.3 Union-Find-class

```
class DisjointSet
{
    // put this in main()
    //vector<int> univ;
    //for (int i = 1; i <= n; i++) univ.push_back(i);
    //DisjointSet ds;
    //ds.makeSet(univ);

    unordered_map<int, int> parent;
    unordered_map<int, int> rank;
    unordered_map<int, int> members;

public:
    void makeSet(vector<int> const &universe)
    {
        for (int i: universe)
        {
            parent[i] = i;
            rank[i] = 0;
            members[i] = 1;
        }
    }

    int Find(int k)
    {
        if (parent[k] != k)
        {
            parent[k] = Find(parent[k]);
        }

        return parent[k];
```

```
  }

  void Union(int a, int b)
  {
    int x = Find(a);
    int y = Find(b);

    if (x == y) {
      return;
    }

    if (rank[x] > rank[y]) {
      parent[y] = x;
      members[x] += members[y];
    }
    else if (rank[x] < rank[y]) {
      parent[x] = y;
      members[y] += members[x];
    }
    else {
      parent[x] = y;
      rank[y]++;
      members[y] += members[x];
    }
  }

  int GetMembers(int a)
  {
    // get the number of members of the disjoint set
       where a is included
    int x = Find(a);
    return members[x];
  }
};
```

2 DP

2.1 Longest Common Subsequence

```
def lcs(X, Y, m, n):
  L = [[0 for i in range(n+1)] for j in range(m+1)]
  for i in range(m+1):
    for j in range(n+1):
      if i == 0 or j == 0:
        L[i][j] = 0
      elif X[i-1] == Y[j-1]:
        L[i][j] = L[i-1][j-1] + 1
      else:
        L[i][j] = max(L[i-1][j], L[i][j-1])
  lcs = ""
  i = m
  j = n
  while i > 0 and j > 0:
    if X[i-1] == Y[j-1]:
      lcs += X[i-1]
      i -= 1
      j -= 1
    elif L[i-1][j] > L[i][j-1]:
      i -= 1
    else:
      j -= 1
  lcs = lcs[::-1]
  print(lcs)
 #test
X = "AGGTAB"
Y = "GXTXAYB"
m = len(X)
n = len(Y)
lcs(X, Y, m, n)
```

2.2 Longest Common Substring

```
def lcsubstring(X: str, Y: str,
                m: int, n: int):
  LCSuff = [[0 for i in range(n + 1)]
             for j in range(m + 1)]
  length = 0
  row, col = 0, 0
  for i in range(m + 1):
    for j in range(n + 1):
      if i == 0 or j == 0:
        LCSuff[i][j] = 0
      else if X[i - 1] == Y[j - 1]:
        LCSuff[i][j] = LCSuff[i - 1][j - 1] + 1
        if length < LCSuff[i][j]:
          length = LCSuff[i][j]
          row = i
          col = j
      else:
        LCSuff[i][j] = 0
  if length == 0:
    print("")
    return
  resultStr = [’0’] * length
  while LCSuff[row][col] != 0:
    length -= 1
    resultStr[length] = X[row - 1]
    row -= 1
    col -= 1
  print(’’.join(resultStr))

lcsubstring(X, Y, m, n)
```

3 Graph

3.1 Bellman-Ford

```
distance = [float(’inf’) for _ in range(n + 1)]
for k in range(1, n):
  for u in range(1, n + 1):
    for v, w in adj[u]:
      distance[v] = min(distance[v], distance[u] + w)
for u in range(1, n + 1):
  for v, w in adj[u]:
    if distance[v] > distance[u] + w:
      report_negative_cycle()
```

3.2 Breadth-First Search

```
distance = [-1 for _ in range(n + 1)]
distance[source] = 0
shortest_path_tree_parent = [-1 for _ in range(n + 1)]
queue = [source]
for u in queue:
  for v in adj[u]:
    if distance[v] == -1:
      distance[v] = distance[u] + 1
      shortest_path_tree_parent[v] = u
      queue.append(v)
```

3.3 Depth-First Search

```
component_index = [-1 for _ in range(n + 1)]
def dfs(u, c):
  component_index[u] = c
  for v in adj[u]:
    if component_index[v] == -1:
      dfs(v, c)

num_components = 0
for u in range(1, n + 1):
    if component_index[u] == -1:
        dfs(u, num_components)
        num_components += 1
```

3.4 Dijkstra

```
import heapq
distance = [float(’inf’) for _ in range(n + 1)]
distance[source] = 0
shortest_path_tree_parent = [-1 for _ in range(n + 1)]
queue = [(distance[source], source)]
done = [False for _ in range(n + 1)]
while len(queue) > 0:
    _, u = heapq.heappop(queue)
    if not done[u]:
        for v, w in adj[u]:
            if distance[v] > distance[u] + w:
                distance[v] = distance[u] + w
                shortest_path_tree_parent[v] = u
                heapq.heappush((distance[v], v))
        done[u] = True
```

3.5 Kruskal

```
void kruskal(vector<pair<ll, pair<ll, ll>>> &res){
    // res == minimum spanning tree vector
    // needs DisjointSet class
    DisJointSet ds;
    vector<int> univ;
    for (int i = 1; i <= n; i++)
        univ.push_back(i);
    ds.makeSet(univ);
    // edges == vector of edges, vector< weight , uv >
    // edges should be sorted.
    for (auto edge : edges){
        int u = edge.second.first;
        int v = edge.second.second;
        if (ds.hasCycle(u, v))
            continue;
        ds.Union(u, v);
        res.push_back(edge);
    }
}
```

3.6 Prim

```
void prim(int start, vector<pair<ll, pair<ll, ll>>> &res){
    // res == minimum spanning tree vector
    priority_queue<pair<ll, pair<ll, ll>>> pq;
    vector<bool> vis(n+1, false);
    vis[start] = true;

    for (auto &[v, w] : graph[start]){
        pq.push({w, {start, v}});
    }

    while (!pq.empty()){
        auto edge = pq.top();
        pq.pop();
        ll u = edge.second.second;
        if (vis[u]) continue;
        vis[u] = true;
        res.push_back(edge);
        for (auto &[v, w] : graph[u])
          if (!vis[v]) pq.push({w, {u, v}});
    }
}
```

3.7 Topological Sort

```
visited = [False for _ in range(n + 1)]
in_dfs_stack = [False for _ in range(n + 1)]
topologically_sorted = []

def toposort(u):
    in_dfs_stack[u] = True
    visited[u] = True
    for v in rev_adj[u]:
        if in_dfs_stack[v]:
            report_cycle()
        elif not visited[v]:
            toposort(v)
    topologically_sorted.append(u)
    in_dfs_stack[u] = False

for u in range(1, n + 1):
    if not visited[u]:
        toposort(u)
```

4 Max Flow

4.1 Convex Hull

```
#finds the smallest convex that contains all the given
     points in O(n^2)
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
def Left_index(points):
    minn = 0
    for i in range(1,len(points)):
        if points[i].x < points[minn].x:
            minn = i
        elif points[i].x == points[minn].x:
            if points[i].y > points[minn].y:
                minn = i
    return minn
def orientation(p, q, r):
    val = (q.y - p.y) * (r.x - q.x) - \
          (q.x - p.x) * (r.y - q.y)
    if val == 0:
        return 0
    elif val > 0:
        return 1
    else:
        return 2
def convexHull(points, n):
    l = Left_index(points)
    hull = []
    p = l
    q = 0
    while(True):
        hull.append(p)
        q = (p + 1) % n
        for i in range(n):
            if(orientation(points[p],
                          points[i], points[q]) == 2):
                q = i
        p = q
        if(p == l):
            break
    for each in hull:
        print(points[each].x, points[each].y)
#test
points = []
points.append(Point(0, 3))
points.append(Point(2, 2))
points.append(Point(1, 1))
points.append(Point(2, 1))
points.append(Point(3, 0))
points.append(Point(0, 0))
points.append(Point(3, 3))
convexHull(points, len(points))
#return all points of the convex
```

4.2 Maximum Bipartite Matching

```
#Uses Hopcroft-Karp Algorithm
class GFG:
    def __init__(self,graph):
        self.graph = graph
        self.ppl = len(graph)
        self.jobs = len(graph[0])
    def bpm(self, u, matchR, seen):
        for v in range(self.jobs):
            if self.graph[u][v] and seen[v] == False:
                seen[v] = True
                if matchR[v] == -1 or self.bpm(matchR[v],
                     matchR, seen):
                    matchR[v] = u
                    return True
        return False
    def maxBPM(self):
        matchR = [-1] * self.jobs
        result = 0
        for i in range(self.ppl):
            seen = [False] * self.jobs
            if self.bpm(i, matchR, seen):
                result += 1
        return result
#test
bpGraph =[[0, 1, 1, 0, 0, 0],
          [1, 0, 0, 1, 0, 0],
          [0, 0, 1, 0, 0, 0],
          [0, 0, 1, 1, 0, 0],
          [0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 1]]
g = GFG(bpGraph)
print(g.maxBPM())
```

5 Miscellaneous

5.1 AA CPP Template

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;

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

    return 0;
}
```

5.2 Bitmasking

```
#Given an array of n integers and an integer x, how many
     subsets
#have a sum equal to x?
def subset_sum(a,n,x):
  ctr = 0
  for mask in range(1<<n): # for each subset
    total = 0
    for i in range(n):
      if mask&(1<<i): # include ith element?
        total += a[i]
      if total == x:
        ctr += 1

  return ctr
```

5.3 KMP Algorithm

```
#Used for Pattern Searching, runs in O(N)
def KMPSearch(pat, txt):
    M = len(pat)
    N = len(txt)
    lps = [0]*M
    j = 0 # index for pat[]
    computeLPSArray(pat, M, lps)
    i = 0 # index for txt[]
    while (N - i) >= (M - j):
        if pat[j] == txt[i]:
            i += 1
            j += 1
        if j == M:
            print("Found pattern at index " + str(i-j))
            j = lps[j-1]
        elif i < N and pat[j] != txt[i]:
            if j != 0:
                j = lps[j-1]
            else:
                i += 1
def computeLPSArray(pat, M, lps):
    lps[0] = 0
    i = 1
    while i < M:
        if pat[i] == pat[len]:
            len += 1
            lps[i] = len
            i += 1
        else:
            if len != 0:
                len = lps[len-1]
            else:
                lps[i] = 0
                i += 1
txt = "ABABDABACDABABCABAB"
pat = "ABABCABAB"
KMPSearch(pat, txt)
#should return pattern at index 10
```

5.4 Recursive Backtracking

```
void rec(string s){
    // n and ans are global variables
    // generate decimals up to n with 3, 5, 7 as its digits
    if (s != "" && stoll(s) > n) return;
    int a=0,b=0,c=0;
    for (auto d : s){
        if (d == ’3’) a++;
        else if (d == ’5’) b++;
        else if (d == ’7’) c++;
    }
    if (a && b && c) ans += 1;
    rec(s + ’3’);
    rec(s + ’5’);
    rec(s + ’7’);
}
```

6 Number Theory

6.1 Prime Sieve

```
def generate_is_prime_sieve_opt(N):
    is_prime = [True] * (N+1)
    is_prime[0] = is_prime[1] = False
    p = 2
    while p*p <= N:
        if is_prime[p]:
            for n in range(p*p, N+1, p):
                is_prime[n] = False
        p += 1
    return is_prime
is_prime = generate_is_prime_sieve_opt(int(input()))
```

7 Search

7.1 Binary Search

```
from bisect import bisect_left
def BinarySearch(a, x):
    i = bisect_left(a, x)
    if i != len(a) and a[i] == x:
        return i
    else:
        return -1
a  = [1, 2, 4, 4, 8]
x = int(4)
res = BinarySearch(a, x)
if res == -1:
    #absent
else:
    #Prints first occurence, returns 2
```

8 zPast Codes

8.1 Christmas

```
#recursion
n,x= [int(x) for x in input().split()]
layer = [1]
patty = [1]
for _ in range(1,51):
    layer.append(layer[_-1]*2 + 3)
    patty.append(patty[_-1]*2 + 1)
def eat(n, x):
    if n == 0:
        if x > 0:
            return 1
        else:
            return 0
    #check if in the lower half
    elif x < (layer[n]+1)//2:
        #eat the first bun then do the next recursion for the
              lowerhalf
        return eat(n-1, x-1)
    #check if in the middle
    elif x == (layer[n]+1)//2:
        return patty[n-1] + 1
    #check if in the upperhalf
    elif x > (layer[n]+1)//2:
        #do the recursion for the upperhalf then take the
             lowerhalf as eaten
        return eat(n-1, x-(layer[n]+1)//2) + patty[n-1] + 1
answer = eat(n, x)
print(answer)
```

8.2 Fractals

```
'Given a pattern, recreate the pattern recursively similar
     to how snowflakes works'
n, k = [int(x) for x in input().split()]
fractal = ['' for x in range(n)]
for x in range(n):
    fractal[x] += input()
def fractalize(orig, new, n, size):
    new_fractal = ['' for x in range(n*size)]
    for x in range(size):
        for y in range(size):
            if new[x][y] == ".":
                for z in range(n):
                    new_fractal[n*x+z] += orig[z]
            else:
                for z in range(n):
                    new_fractal[n*x+z] += n*'*'
    return new_fractal
ans = fractal.copy()
for x in range(k-1):
    ans = fractalize(fractal, ans, n, n**(x+1))
for x in ans:
    print(''.join(x))
```