Team notebook

Polytech Nice Sophia - Polytech Nice A

January 20, 2020

Contents

1 dp

1.1 Longest Increasing subsequence (LIS)

```
def pgss(L):
    """
    L: a list of integers
    return: the longest increasing subsequence of L
    """
        S=[(0,L[0])]
        hist = {(0,L[0]) : None}

        for t in enumerate(L[1:]):
                a,b = 0,len(S)-1
                while a<=b:
                        p = (a+b)//2
                        if S[p][1]<t[1]:
                                a=p+1
                        else:
                                b=p-1

                if a>=len(S):
                        S.append(t)
                else:
                        S[a] = t
                hist[t] = S[a-1] if a>0 else None

        v = S[-1]
        L = []
        while v is not None:
                L.append(v[1])
                v=hist[v]
        return L[::-1]
```

2 geometry

2.1 convex hull

```
from functools import cmp_to_key

    def compute_convex_hull(pts):
        """
        pts: a list of tuple coordinates
        return: the list of the convex hull of all the points
              under a list of coordinates
        """
        def area(c1, c2, c3):
            return (c2[0] - c1[0]) * (c3[1] - c1[1]) - (c3[0] -
                 c1[0]) * (c2[1] - c1[1])

        def is_inside(c1, c2, c3):
            a = area(c1, c2, c3)
            if a == 0:
                return max(c1[0], c2[0]) >= c3[0] >= min(c1[0],
                     c2[0])
            else:
                return a > 0

        def dist(c1,c2):
            return ((c2[0]-c1[0])**2)+((c2[1]-c1[1])**2)

        pivot=min(pts, key=cmp_to_key(lambda c1,c2: c1[0]-c2[0]
              if c1[1]==c2[1]
            else c1[1]-c2[1]))
        pts.remove(pivot)
        vertices=sorted(pts,key=cmp_to_key(lambda c1,c2:
              area(pivot,c2,c1)))
        hull = [pivot]
        yield hull
        idx=0
        for v in vertices:
            if len(hull) < 2:
                hull.append(v)
            else:
                ar = area(hull[-2], hull[-1], v)
                if ar==0:
                    hull[-1]=max(v,hull[-1],key=lambda
                          x:dist(hull[-2],x))
                    continue
                while ar<=0 and len(hull)>1:
                    hull.pop()
                    ar=area(hull[-2], hull[-1], v)
                hull.append(v)
            yield hull
        hull.append(hull[0])
        return hull
```

3 graph

3.1 bellman ford

```
def bellman_ford(n_edge, vertices, edges):

        lbl = {v:0 for v in vertices}

        for k in range(1,n_edge):
                for v in vertices:
                        d=lbl[v]
                        for n in edges[v]:
                                d=min(d,lbl[n]+edges[v][n])
                        lbl[v]=d

        return -min(lbl.values())
```

3.2 flow dinic CPP

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u),
          cap(cap) {}
};
```

```
struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size();
             cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap -
                 edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap -
                 edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

3.3 flow dinic PYTHON

```
#From algorithmie efficace
    from collections import deque

    def dinic(graph, capacity, source, target):
        assert (source!=target)
        add_reverse_arcs(graph, capacity)
        Q=deque()
        total=0
        n=len(graph)
        flow=[[0]*n for u in range(n)]
        while True:
                Q.appendleft(source)
                lev = [None]*n
                lev[source]=0
                while Q:
                        u=Q.pop()
                        for v in graph[u]:
                                if lev[v] is None and
                                     capacity[u][v] > flow[u][v]:
                                  lev[v] = lev[u] + 1
                                  Q.appendleft(v)
                if lev[target] is None:
                        return flow, total
                #UB = borne sup
                UB = sum(capacity[source][v] for v in
                     graph[source]) - total
                total+= _dinic_step(graph, capacity, lev, flow,
                     source, target, UB)

    def _dinic_step(graph, capacity, lev, flow, u, target,
         limit):
        if limit <=0:
                return 0
        if u == target:
                return limit
        val = 0
        for v in graph[u]:
                residuel = capacity[u][v] - flow[u][v]
                if lev[v] == lev[u] + 1 and residuel > 0:
                        z=min(limit, residuel)
                        aug = _dinic_step(graph, capacity, lev,
                             flow, v, target, z)
                        flow[u][v]+=aug
                        flow[v][u]-=aug
                        val+=aug
                        limit-=aug
                if val==0:
                        lev[u]=None #Sommet non frachissable
                             enlever
                return val
```

3.4 kruskal

```
def kruskal(edges):
        """
        edges: is like [(n1,n2,weight),...]
        return: the total length of the minimum spanning tree
        """

        parent = dict()
        rank = dict()

        def make_set(x):
                parent[x]=x
                rank[x]=0

        def union(x,y):
                gx,gy = find(x), find(y)
                if gx != gy:
                        if rank[gx] < rank[gy]:
                                parent[gx] = gy
                        else:
                                parent[gy] = gx
                                if rank[gx] == rank[gy]:
                                        rank[gx]+=1

        def find(x):
                if parent[x]!=x:
                        parent[x] = find(parent[x])
                return parent[x]

        for n1,n2,d in edges:
                make_set(n1)
                make_set(n2)

        edges = sorted(edges, key=lambda x:x[2])

        mst_l=0
        for n1,n2,d in edges:
                if find(n1) != find(n2):
                        mst_l+=d
                        union(n1,n2)
        return mst_l
```