Recitation 10 (HW9)

Online: Xinyi Zhao xz2833@nyu.edu

GCASL 461: Yifan Jin yj2063@nyu.edu

New York University

Basic Algorithms (CSCI-UA.0310-005)

Problem 1

Problem 1 (20 points)

Let G be a directed graph. After running DFS algorithm on G, the vertex v of G ended up in a DFS tree containing only v, even though v has both incoming and outgoing edges in G. Fully explain how this happened.

Problem 1

Recall DFS(G)

Preprocessing

for every vertex v in G:

(i) parent : v.parent := void → construct DFS tree

(ii) label : v.l → undiscovered, active, seen → avoid repetitions & avoid indefinite running

v.l := undiscovered

(iii) v.d : discovery time of v

(iv) v.f : seen time of v

useful for DFS applications (edge clarification, finding cycles, topological sorting)

Problem 1

Recall DFS(G)

DFS(G):

input: "directed" graph G

output: all vertices in G

DFS(G)

vertex set of G

(1) for each vertex s in V(G)

if (s.l == undiscovered)

DFS(G, s)

DFS(G, s) (recursive implementation)

O(1) time: time += 1, s.d = time, s.l = active — optional for applications

for u in V[s]

O(1) time:

if u.l == undiscovered

u.parent = s

DFS(G, u)

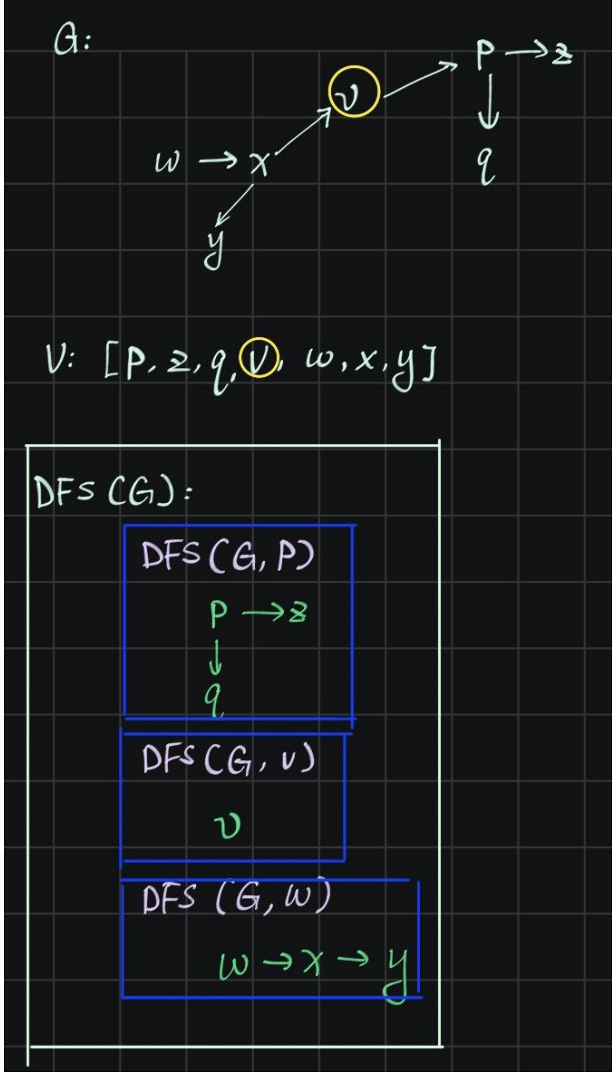
O(1) time:

s.l = seen

time += 1 — optional for applications

s.f = time

Problem 1



Problem 1

- v has incoming and outgoing edges:

$$G_1 \to v \to G_2$$

- DFS (G):

a DFS tree containing only v

$\Rightarrow$ ① $v \to G_2$

all the vertices in G_2 are visited before visiting v

② $G_1 \to v$

v is visited before all the vertices in G_1.

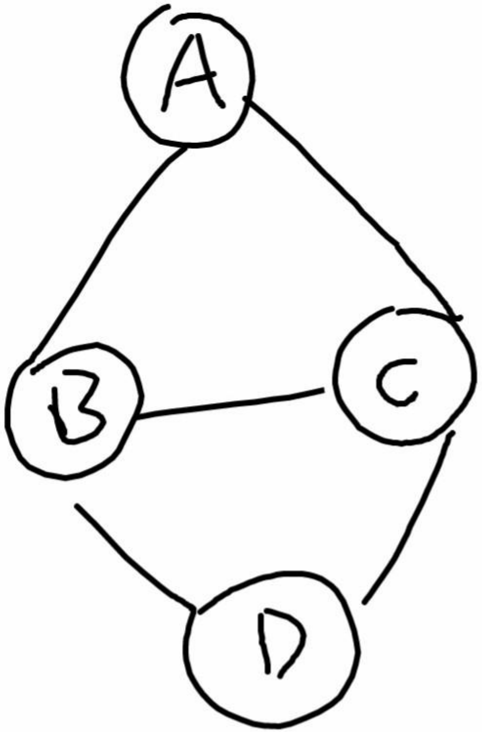
Problem 2

Problem 2 (25 points)

Recall that a cycle in an undirected graph is a sequence of distinct vertices $(v_1, v_2, \ldots, v_k)$ with $k \geq 3$ such that the edges $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}$, and $\{v_k, v_1\}$, all belong to the graph.
Given an undirected connected graph $G = (V, E)$, develop an algorithm that determines whether G has a cycle. Your algorithm must run in $O(|V| + |E|)$ time.

Problem 2

If there is a cycle, what will happen when we run DFS?

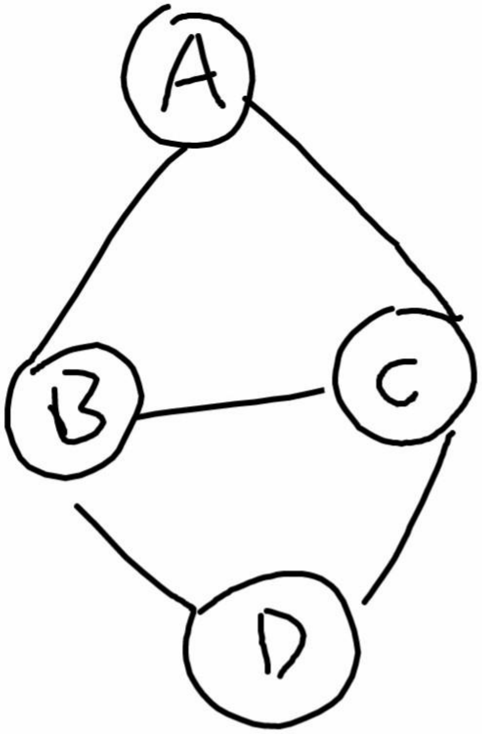


A->B->C->A

A->B->D->C->A

Problem 2

If there is a cycle, what will happen when we run DFS?



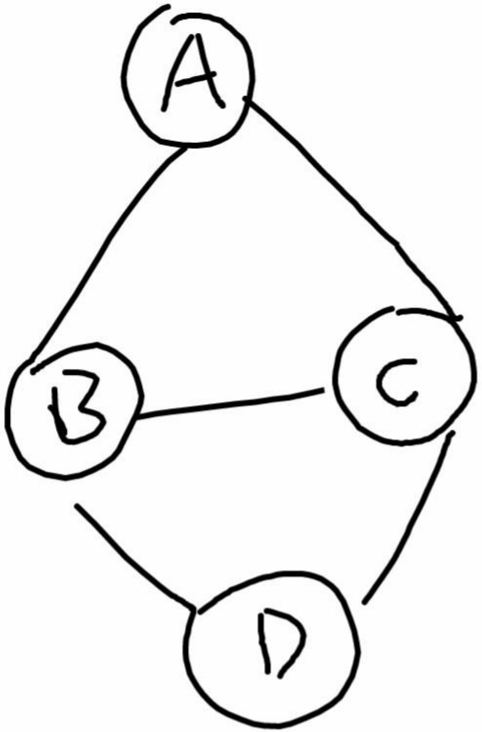
A->B->C->A

A->B->D->C->A

In both cases, A is **discovered** when the current node is C.

Problem 2

If there is a cycle, what will happen when we run DFS?



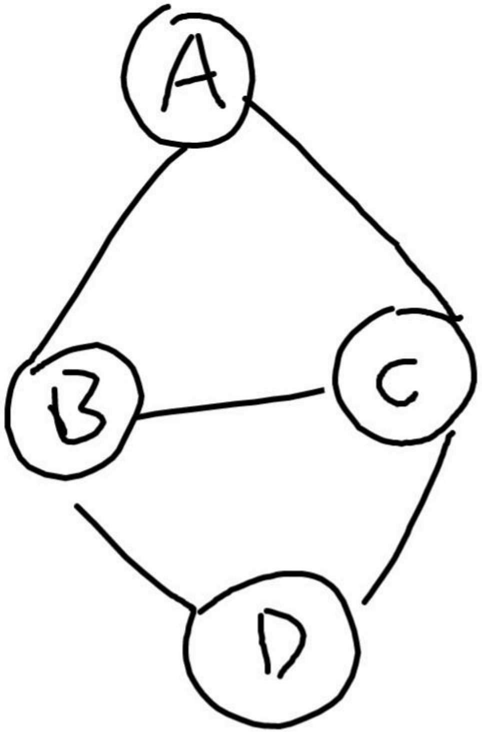
A->B->A (not a cycle)

What if the traversal order is like above?

A is discovered when we are at B.

Problem 2

If there is a cycle, what will happen when we run DFS?



A->B->A (not a cycle)

What if the traversal order is like above?

A is discovered when we are at B.

NOT the parent node

Problem 2

DFS(G, s) (recursive implementation)

O(1) time
time += 1
s.d = time
s.ℓ = active
} optional for applications

for u in V[s]

O(1) time
if u.ℓ == undiscovered
u.parent = s
DFS(G, u)

Else if s.parent != u:
Report a cycle is found

O(1) time
s.ℓ = seen
time += 1
s.f = time
} optional for applications

crossing edge

Problem 2

Pseudocode :

```
has_cycle (G):
    for s in V(G):
        if (s.l == undiscovered):
            if ( DFS (G, s) == true):
                return true
    return false
```

init call: has-cycle(G)
TC = O(|V|+|E|)

```
DFS (G, s):
    s.l = active
    for u in V[s]:
        if ( u.l == undiscovered):
            u.parent = s
            if ( DFS (G, u)):
                return true
        else if ( s.parent != u):
            return true
    s.l = seen
    return false
```

Problem 3

Problem 3 (15+15 points)

Provide counterexamples to each of the following statements.

(a) If a directed graph G contains a path from u to v, and if $u.d < v.d$ in a depth-first search of G, then v is a descendant of u in the depth-first forest produced.
Note that a forest is a union of some disjoint trees.

(b) If a directed graph G contains a path from u to v, then any depth-first search on G must result in $v.d \leq u.f$.

Problem 3

(a) If a directed graph G contains a path from u to v, and if $u.d < v.d$ in a depth-first search of G, then v is a descendant of u in the depth-first forest produced.
Note that a forest is a union of some disjoint trees.

Counterexample: there is a path from u to v and u.d < v.d, v is **NOT** a descendant of u.

Problem 3

(a) If a directed graph G contains a path from u to v, and if $u.d < v.d$ in a depth-first search of G, then v is a descendant of u in the depth-first forest produced.
Note that a forest is a union of some disjoint trees.

Counterexample: there is a path from u to v and u.d < v.d, v is **NOT** a descendant of u.

Idea: **The path from u to v should not be traversed in DFS, otherwise v must be a descendant of u.**

How?

Problem 3

(a) If a directed graph G contains a path from u to v, and if $u.d < v.d$ in a depth-first search of G, then v is a descendant of u in the depth-first forest produced.
Note that a forest is a union of some disjoint trees.

Counterexample: there is a path from u to v and u.d < v.d, v is **NOT** a descendant of u.

Idea: **The path from u to v should not be traversed in DFS, otherwise v must be a descendant of u.**

How?

Let one node in the path discovered already.

Problem 3

(a) If a directed graph G contains a path from u to v, and if $u.d < v.d$ in a depth-first search of G, then v is a descendant of u in the depth-first forest produced.
Note that a forest is a union of some disjoint trees.

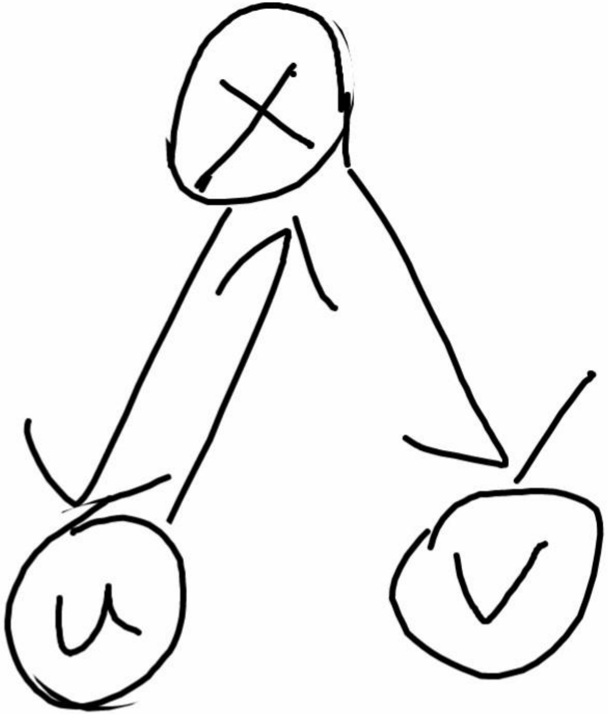
Counterexample: there is a path from u to v and u.d

Idea: **The path from u to v should not be traversed descendant of u.**

How?

Let one node in the path discovered already.

Starting from x, visit u first,
then return to x, visit v



Problem 3

(b) If a directed graph G contains a path from u to v, then any depth-first search on G must result in $v.d \leq u.f$.

Counterexample: a path from u to v, there exists a DFS resulting in v.d > u.f

Problem 3

(b) If a directed graph G contains a path from u to v, then any depth-first search on G must result in $v.d \leq u.f$.

Counterexample: a path from u to v, there exists a DFS resulting in v.d > u.f

Idea: **u must be totally discovered and returned before visiting v.**

The path from u to v should not be traversed in DFS

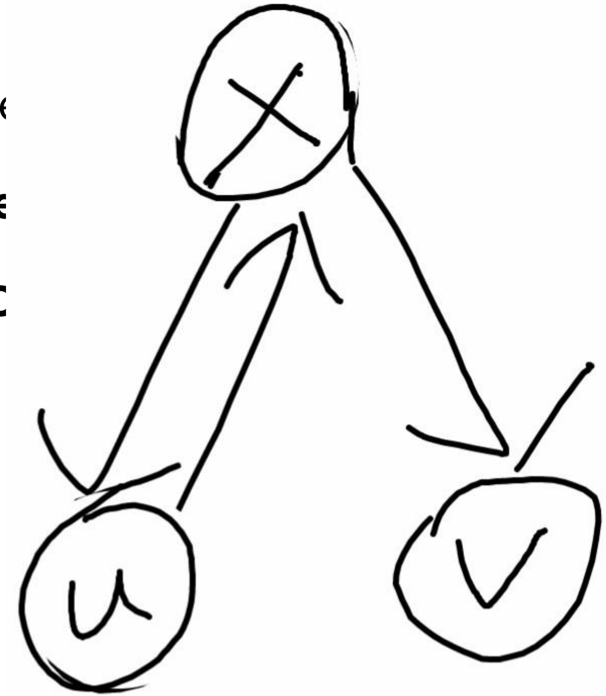
Problem 3

(b) If a directed graph G contains a path from u to v, then any depth-first search on G must result in $v.d \leq u.f$.

Counterexample: a path from u to v, there exists a DFS re

Idea: **u must be totally discovered and returned before**

The path from u to v should not be traversed in D



Same as before.

Starting from x, visit u first,
then return to x, visit v

Problem 4

Problem 4 (25 points)

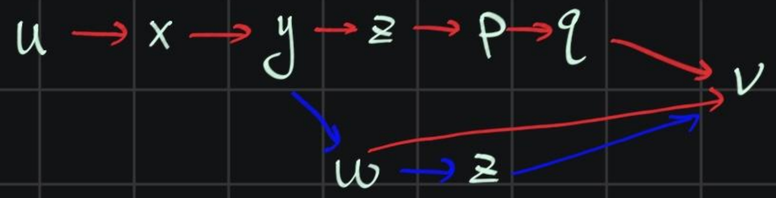
Given a directed graph and two vertices u, v in the graph, the function P finds the shortest path from u to v in that graph. We have access to the function P and want to use it to solve the following problem.

Consider the directed graph G such that each edge of G is colored by red or blue. Given two vertices s, t in G, we want to find the shortest path from s to t in G which is *color-separable*. In other words, the path must first consist of some red edges (possibly none of them) followed by some blue edges (possibly none of them). Thus, once we encounter a blue edge in the path, the following edges in the path must be also blue.

Develop an $O(|V| + |E|)$-time algorithm to find the shortest color-separable path from s to t in G. Your algorithm must use the function P once on a carefully constructed graph. Note that you should not use BFS or explore the graph yourself. Instead, you must call the function P.
Justify why the running time of your algorithm satisfies the required bound.

Hint: First, consider the graph with only the red edges of G. Then, restrict yourself to the graph with only the blue edges of G. Can you combine these two graphs somehow to construct a new graph and solve the problem?

Problem 4



paths from u to v:

① $u \to x \to y \to z \to p \to q \to v$ $d=6$, color-seperate

② $u \to x \to y \to w \to v$ $d=4$, not color-seperate

③ $u \to x \to y \to w \to z \to v$ $d=5$, color-seperate $\Rightarrow$ shortest color-seperate path

Problem 4

What ~~是~~ are the difficulties in this problem?

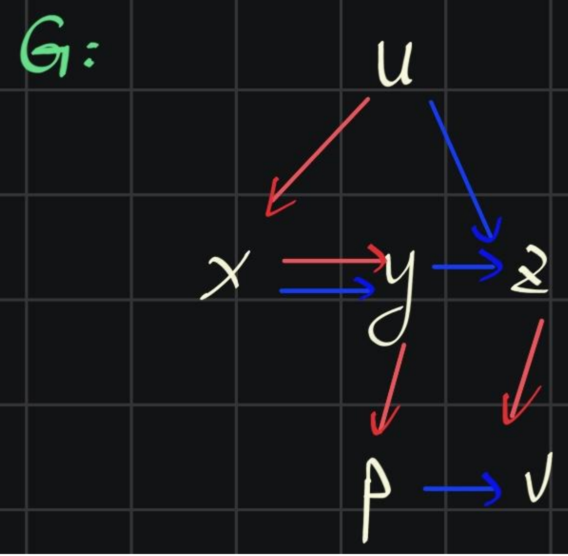
① what is a color - seperate path?

- red
- blue
- red — blue

② How to call P once to find the shortest color-ceperate path??

— Construct a new graph, such that the edges are well-organized in case of colors.

Problem 4



shortest color-seperate path:

$$\boxed{u \rightarrow x \rightarrow y \rightarrow p \rightarrow v}$$

$d = 4$

Problem 4

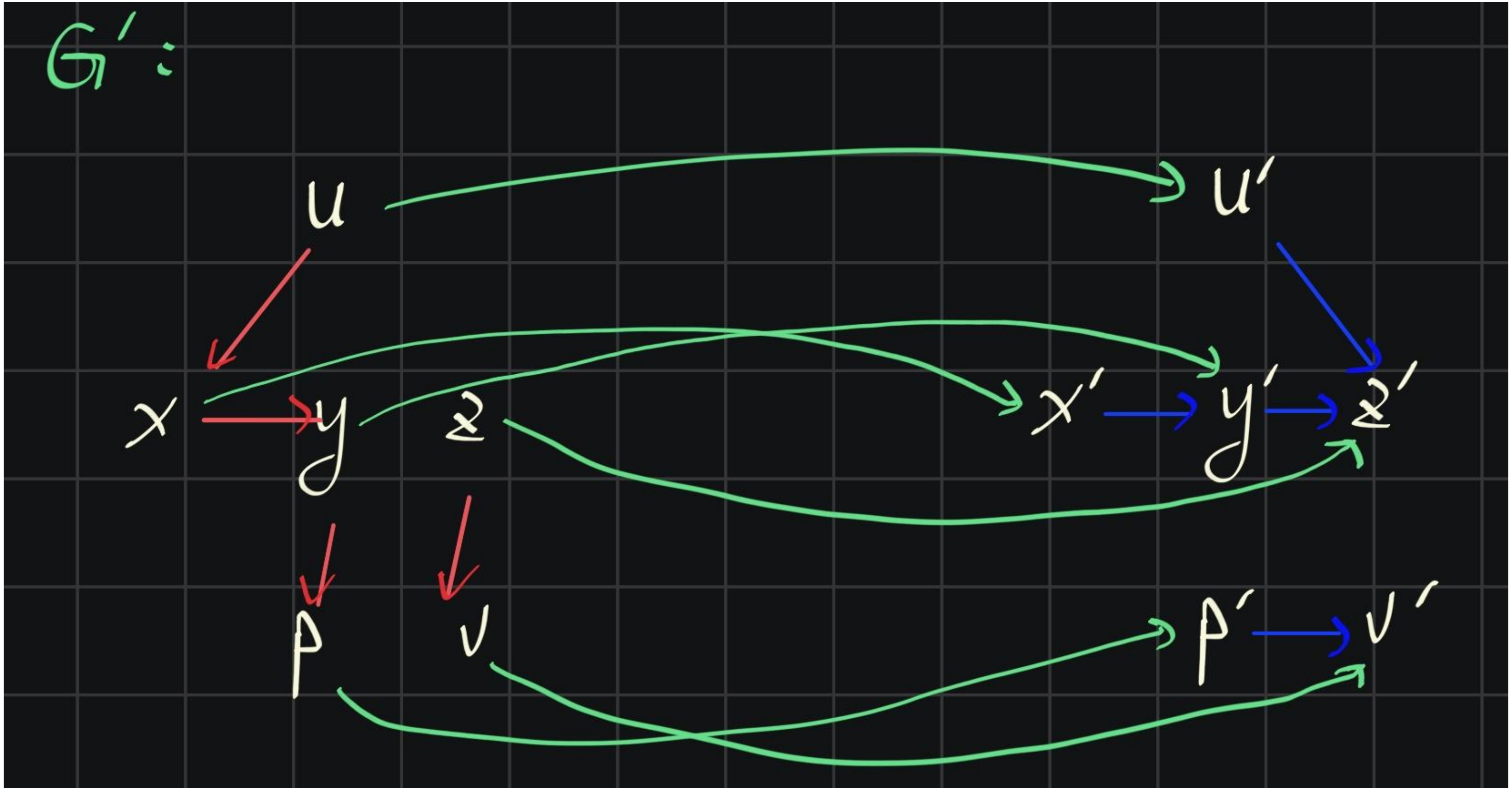
R:

$$\begin{tikzcd}
 & u \arrow[dl] & \\
x \arrow[r] & y \arrow[d] & z \arrow[d] \\
 & p & v
\end{tikzcd}$$

B:

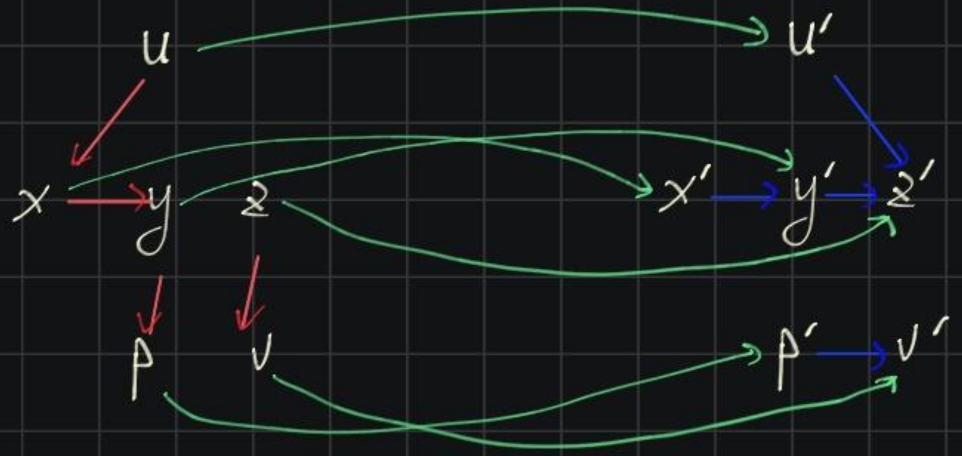
$$\begin{tikzcd}
 & u' \arrow[dr] & \\
x' \arrow[r] & y' \arrow[r] & z' \\
 & p' \arrow[r] & v'
\end{tikzcd}$$

Problem 4



Problem 4

G':



$P(G', u, v')$: $u \to x \to y \to p \xrightarrow{0} p' \to v'$, $d = 4$

Problem 4

Algorithm:

1. Copy each vertex s from G, to a new graph R.
2. Copy each vertex s as s' from G, to a new graph B.
3. Copy red edges from G to R.
4. Copy blue edges from G to B.
5. Construct a new graph G': add one edge with weight 0 from each s in R to s' in B. $(s \to s')$
6. Call $P(G', u, v')$

Problem 4

Algorithm:

$O(|V|+|E|)$

1. Copy each vertex s from G, to a new graph R. $O(|V|)$
2. Copy each vertex s as s' from G, to a new graph B. $O(|V|)$
3. Copy red edges from G to R. $O(|E|)$
4. Copy blue edges from G to B. $O(|E|)$
5. Construct a new graph G': add one edge with weight 0 from each s in R to s' in B. $(s \rightarrow s')$ $O(|V|)$
6. Call $\boxed{P(G', u, v')}$

Problem 4 (HW8)

We want to develop a divide and conquer-based algorithm to find the convex hull of n given points.

(a) First, given two convex polygons P and Q which have n points in total, develop an $O(n)$-time algorithm to find the convex hull of their union.
You have to fully explain your algorithm. You do NOT need to write the pseudo-code of your algorithm.

Problem 4 (HW8)

We want to develop a divide and conquer-based algorithm to find the convex hull of n given points.

(a) First, given two convex polygons P and Q which have n points in total, develop an $O(n)$-time algorithm to find the convex hull of their union.
You have to fully explain your algorithm. You do NOT need to write the pseudo-code of your algorithm.

Recall that the running time of Graham's Scan being O(nlogn) is totally because the sorting algorithm is O(nlogn).

All remaining part is O(n).

Problem 4 (HW8)

We want to develop a divide and conquer-based algorithm to find the convex hull of n given points.

(a) First, given two convex polygons P and Q which have n points in total, develop an $O(n)$-time algorithm to find the convex hull of their union.
You have to fully explain your algorithm. You do NOT need to write the pseudo-code of your algorithm.

Another solution: **Copy the idea of Merge function in Merge-Sort.**

Why could we do like that?

Problem 4 (HW8)

Merge: Merge two sorted arrays into one sorted array, in O(n) time

Here: Union two convex hulls into one convex hull, in O(n) time

Convex hull is a set of **'sorted'** edges (in counter-clockwise order).

The order is based on the angle from the chosen P_0.

Problem 4 (HW8)

Recall Merge function in merge sort:

```
MERGE(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
create arrays L[1..n1 + 1] and R[1..n2 + 1]
for i ← 1 to n1
        do L[i] ← A[p + i − 1]
for j ← 1 to n2
        do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
        do if L[i] ≤ R[j]
              then A[k] ← L[i]
                   i ← i + 1
              else A[k] ← R[j]
                   j ← j + 1
```

Problem 4 (HW8)

```
MERGE(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
create arrays L[1 .. n1 + 1] and R[1 .. n2 + 1]
for i ← 1 to n1
      do L[i] ← A[p + i − 1]
for j ← 1 to n2
      do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
      do if L[i] ≤ R[j]
            then A[k] ← L[i]
                  i ← i + 1
            else A[k] ← R[j]
                  j ← j + 1
```


Comparison is based on angle from P_0

Stack operation

Problem 4 (HW8)

```
MERGE(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
create arrays L[1..n1 + 1] and R[1..n2 + 1]
for i ← 1 to n1
    do L[i] ← A[p + i − 1]
for j ← 1 to n2
    do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
    do if L[i] ≤ R[j]
        then A[k] ← L[i]
             i ← i + 1
        else A[k] ← R[j]
             j ← j + 1
```


How to choose a P_0 given two convex hulls?

How to define i,j?

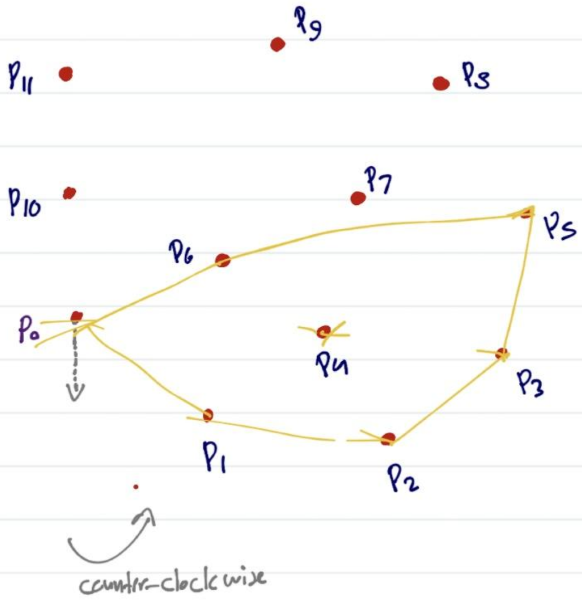
Comparison is based on angle from P_0

Stack operation

Problem 4 (HW8)

Recall Graham's scan

Example



Graham's scan

(1) find the init'l point on the convex hull : extreme corner points

the point with min x-coordinate among all the given points (the leftmost point) and if there is a tie, choose the bottom point among the leftmost points (the one with min y-coordinate) : P_0

(2) Sort out the rest of the points wrt. the angles they form with the vertical half-line emanated downwards from P_0 :

$P_1, \ldots, P_{n-1}$: sorted in counter-clockwise order wrt. P_0

(3) Define an empty stack S :

S.push(P_0), S.push(P_1), S.push(P_2)

(4) for k=3 to n

while the 3 points {S.before-top(), S.top(), P_k} form a clockwise direction

S.pop()

S.push(P_k)

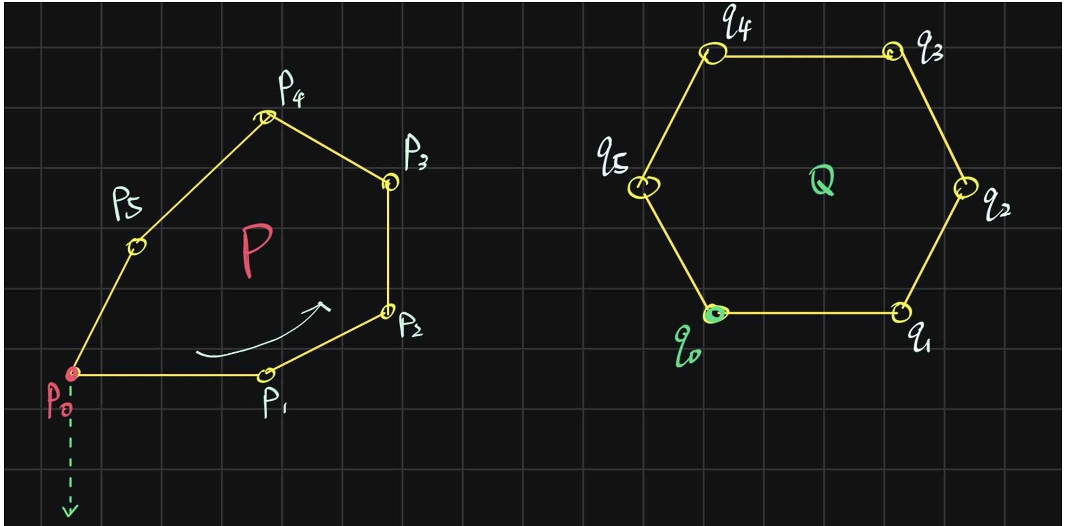
(5) return S

Problem 4 (HW8)

1. Start with the leftmost point among P and Q (find it in linear time). Assume it is p_0 and belongs to P. The rest of the points in P are sorted in the counterclockwise order wrt p_0: p_1,...,p_{n-1}.

Problem 4 (HW8)

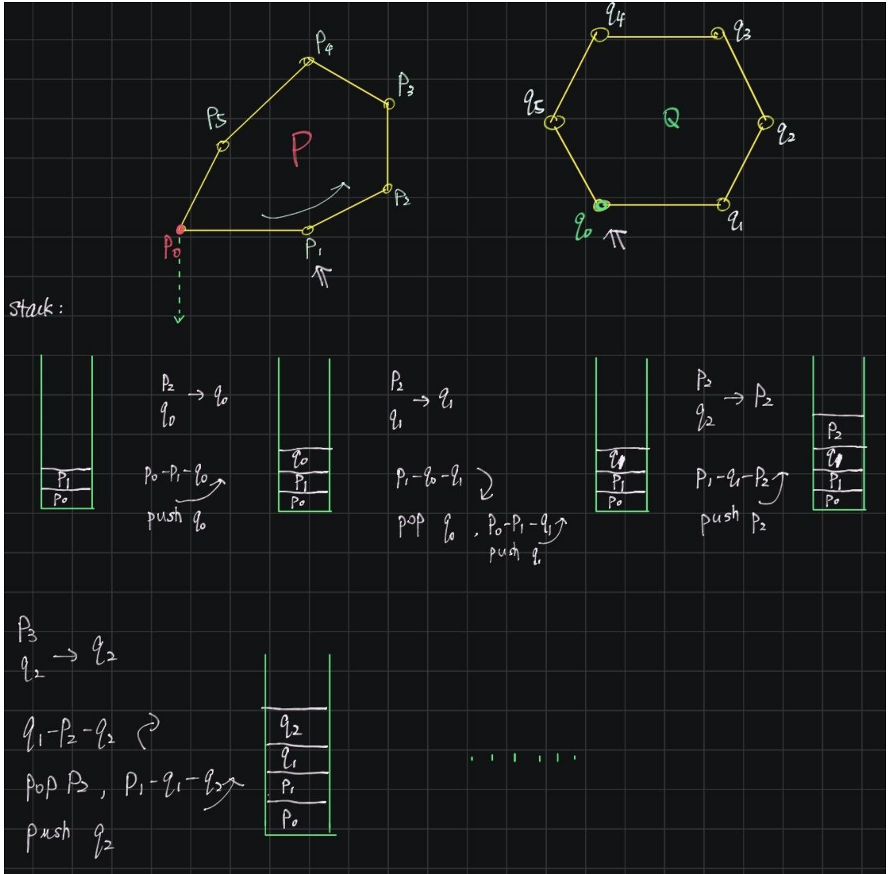
1. Start with the leftmost point among P and Q (find it in linear time). Assume it is p_0 and belongs to P. The rest of the points in P are sorted in the counterclockwise order wrt p_0: p_1,...,p_{n-1}.
2. Find the point in Q which determines the least angle with p_0 (as in Graham's scan) in linear time. Let it be q_0 and the rest of the points of Q in the counterclockwise order be q_1,...,q_{m-1}.



Problem 4 (HW8)

1. Start with the leftmost point among P and Q (find it in linear time). Assume it is p_0 and belongs to P. The rest of the points in P are sorted in the counterclockwise order wrt p_0: $p_1,...,p_{n-1}$.
2. Find the point in Q which determines the least angle with p_0 (as in Graham's scan) in linear time. Let it be q_0 and the rest of the points of Q in the counterclockwise order be $q_1,...,q_{m-1}$.
3. Combine the for-loop of Graham's scan with the idea of the Merge function in Mergesort:
   - Let the first unchosen points of P and Q be p_i and q_j at this step. Among p_i and q_j, take the one which determines a smaller angle with p_0. For example, if it is p_i, perform the stack operations with p_i. Increase $i=i+1$, and go over the for-loop until you are done with all the points of P and Q, i.e., $i=n$ and $j=m$.

Problem 4 (HW8)



Q & A

Thank you