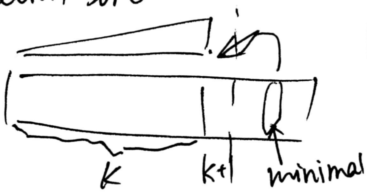
Sorting algorithms

$O(n^2)$ Bubble. Insert. Selection

Selection sort.



$\forall i<k,\ j\geq k.\ a_i \leq a_j$

$\downarrow$

$\forall i\leq k,\ j>k\ \ a_i \leq a_j$

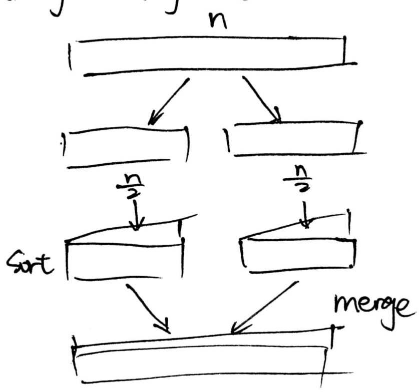
$\downarrow$

$\forall i\leq n,\ j>n\ \ a_i \leq a_j$

```
Selectionsort (a):
  n=|a|
  for k=0 ~~to~~ n-1
    min_ind=k
    for j=k+1 to n-1
      if a[j] < a[min_ind]
        min_ind=j
    swap(a[k], a[min_ind]).
```

Compelexity: $(n-1)+(n-2)+\dots+1 = \frac{n(n-1)}{2}$ $O(n^2)$

$O(n\log n)$ Merge sort.




```
merge_sort(a) <—— O(logn)
  n=|a|  <—— n=1 ~~stop~~ return
  b= a[0 : n/2 - 1]
  c= a[n/2 ... n-1]
  merge_sort(b)
  merge_sort(c)
  a= merge(b,c)
```

```
merge(b,c) <—— O(n)
  n=|b|  m=|c|
  n=|b|  m=|c|
  i=0 j=0
  when i<n and j<m
    if (j=m) or (i<n and b[i]<c[j])
      ~~i~~ res[i+j]=b[i], i++
    else
      res[i+j]=c[j], j++
  return res.
```

$O(n)$ Counting Sort

eg. | 0 | 1 | 1 | 2 | 4 | 1 | 0 | 1 |

$a<b$ and $b<c \Rightarrow a<c$

0 : 2
1 : 4
2 : 1
4 : 1

```
Count_sort(a)          O(n) -> O(n+k)
  for i=0 to n-1
    cnt[a[i]]++
  for i=0 to k    <—— maxium a[i]
    for j=0 to cnt[i]
      print(i)
```

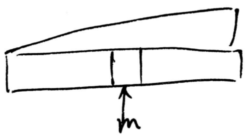
Binary Search.

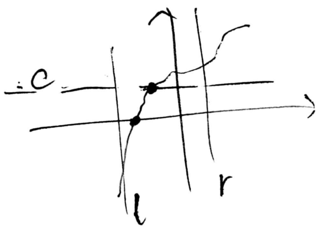
Linear search

if $a[i]=x$

$O(n)$

if the array is already sorted.





Binary Search to find a root of the Function / take the value c

Dynamic Programming.

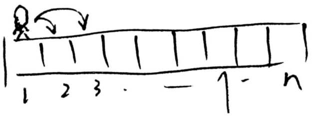
Base $n=0,1$ statement for n.
step $n \to n+1$

eg. $f_0 = f_1 = 1$
$f_n = f_{n-1} + f_{n-2}$

```
fibb(n)
  if n=0 || n=1
    return 1
  return fibb(n-1) + fibb(n-2)
```

斐波那契数列

exponantial



ways[1] = 1   ways[2] = 1
for i = 3 ... n
  ways[i] = ways[i-1] + ways[i-2]

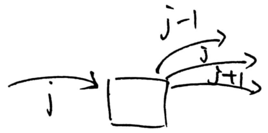
```
f[0] = 1
f[1] = 1
for 2...n
  f[i] = f[i-1] + f[i-2]
```

$\Leftarrow$

```
fib(n)
  if n=0 || n=1
    return 1
  if f[n] = -1
    f[n] = fib(n-1) + fib(n-2).
  return f[n]
```

if some ~~se~~ cells are forbidden

for i = 3 ... n
  ~~ways~~
  if not forbidden
    ways[i] = ways[i-1] + ways[i-2]

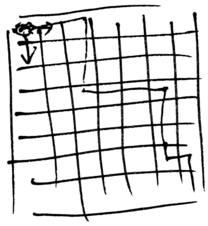


two statement

(i, j)

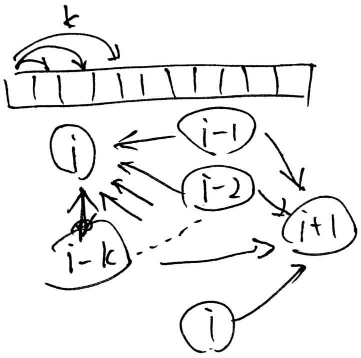
(i-j, j-1)   (i-j, j)   (i-j, j+1)

$O(n^2)$



$O(n \cdot k)$
waste for i
ways[i] = ways[i-1] + ways[i-1] - ways[i-k-1]

res[i] = c[i] + max(res[i-1] ... res[i-k])



statement

(i, j)

(i-1, j)   (i, j-1).

base
$(0,0) \to C_{00}$
$(0,i) \to C_{00} + C_{01} + \cdots C_{0i}$
$(j,0) \to C_{00} + C_{10} + \cdots C_{j0}$

for i = 1 ... n
  for j = 1 ... n
  $\boxed{res[i][j] = C_{ij} + max(res[i-1][j], res[i][j-1])}$

$\Downarrow$ from[i][j]

if res[i-1][j] > res[i][j-1]
  res[i][j] = res[i-1][j] + C_{ij}
  from[i][j] = [i-1, j]
else.   <

Longest increasing ~~ab~~sequence

| a_1 | a_2 | ... | a_n |

dp[i] = LIS that ends in i
dp[i] = 1
  for i = 2 ... n
    for j = 1 ... i-1
$O(n^2)$ if a[i] > a[j]
        dp[i] = max(dp[i], dp[j]+1)

| 3 | 4 | 1 | 2 | 6 | 5 |

find using binary search
$O(n \log n)$

Longest common subsequence

|a_1|a_2| ---- |a_n|

|b_1|b_2| ---- |b_n|

eg. |1|2|5|3|4|

|1|5|2|4|3|

dp[i][j] = LCS a[1 ⋯ ~~i~~] and [1 ⋯ j]

(i, j) ← (i−1, j), (i, j−1), (i−1, j−1) if a[i] = b[j]

```
for i=1 ... n
  for j=1 ... m
    if a[i]=b[j]
      dp[i][j] = max(dp[i][j], dp[i-1][j-1])
    else
      dp[i][j] = max(dp[i-1][j], dp[i][j-1])
```

Levenstein distance (Edit distance)

ababac
babbc

three operations: insert / ~~dete~~ delete / replace char.

eg. ababac / babbc → delete → replace → babbc / babbc   result: 2

(i, j) ← i, j−1 insert
(i, j) ← i−1, j delete
(i, j) ← i−1, j−1 replace

```
for i=1 ... n
  for j=1 ... m
    if (j>i)
      dp[i][j]=min(dp[i][j], dp[i][j-1]+1)   insert
    if (i>j)
      dp[i][j] - - -        delete
    if i>1 && j>1
      if equal : dp[i][j] = min(dp[i][j], dp[i-1][j-1])  (do nothing)
      else : dp[i][j] = min(dp[i][j], dp[i-1][j-1]+1)   replace
```

Graph

G = <V, E>   V: vertex, E: Edge

E = {(v, u) | v, u ∈ V}

1) directed / undirected
2) tree

How to save the graph in computer
1) List of edges   O(|E|)
2) Adjagency Matrix.   O(|V|²)
3) Adjacency List   O(|E|)

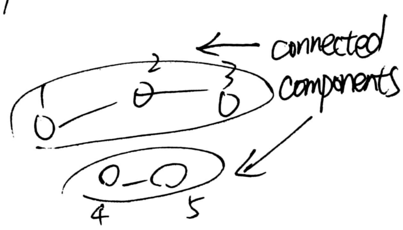
path = $V_1 V_2 \cdots V_k$ | $(V_i V_j) \in E$
Cycle = path, $V_1 = V_k$

Trans "List of edges" to "Adj List"

```
read (n, m)
for i=0 to n-1
  read (a, b)
  a--
  b--
  g[a].append(b)
  g[b].append(a)  ← erase this line for directed graphs
```

1) is graph Connected?
~~D~~BFS (depth first search)

sudo code:
```
dfs(v, visited)
  visited[v] = true
  for u: g[v]
    if not visited[u]
      dfs(u, visited)
```



change visited to color
0 is default.
and add current color.
a new parameter.

2) Count Connected Components

```
cnt
for i=0 to n-1
  if color[i]=0
    cnt++
    dfs(i, color, cnt)
```

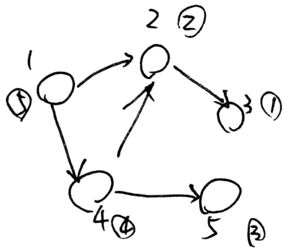
3) check is there a cycle in graph.

```
dfs (v. colour)
  colour[V]=1
  for u:g[V]          if colour[u]=1
  |  if color[u]=0      a cycle is found.
  |    dfs(u. colour)
  color[V]=2
```

```
restor_cycle (v.u.p)
  res=[]
  while v≠u
    res.append(v)
    V=P[V]
  res.append(u)
  reverse (res)
  return res
```

4) topological sort

```
dfs (v. color)
  ...
tout[V]=T, T++
or( ans.append(v))
  quicker.
```



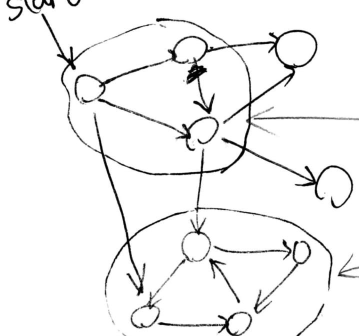
~~Sort~~ Sort tout[V] ↓ decend order

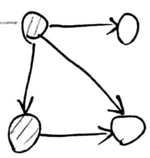
1 4 5 2 3

5) strongly connected components

for each uv
u→v ⟺ v→u

start



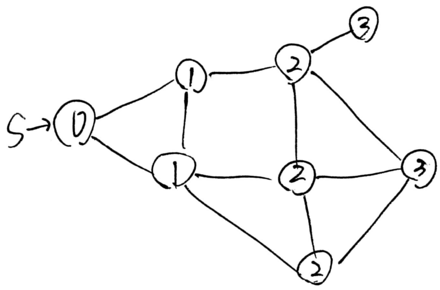


condensation of the left graph (no circle)

1) sort v by tout in circle
2) run connected components dfs on reversed graph in order of I

```
for i=0 to n-1
|  if visited[V]=0
|    topo_sort(g.V.visited)
reverse (ans)
  for i=0 to n-1
    V=ans[i]
    if color[V]=0
      cnt+1
      dfs(reverse g.V.color.cnt)
```

Shortest paths & Breadth First Search



0-1-BFS
(when the edge is 0 and 1)

0-K-BFS
$O(m+nk)$

Dijkstra $O(m\log n)$ $O(n^2+m)$

Ford-Bellman

dp[k][v] – shortest path to v with ~~path~~ length ≤ k

```
dp[k+1]  for k=1...n-1
            for v∈V
              dp[k+1][v]=dp[k][v]
            for v∈V
              for w∈E
                dp[k+1][n]=min(dp[k+1][v], dp[k][v]+w)
```

Floyd

```
d[v][u]
for k=1...n
  for i=1...n
    for j=1...n
      dp[i][j]=min(dp[i][j], dp[i][k]+dp[k][j])
      from[i][j]=k
```

restore(i, j)

~~k~~.

What the hell !!!

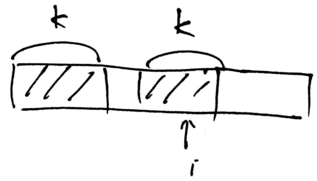
Strings

String: sequence of characters

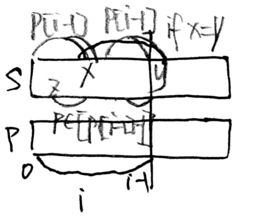
prefix-function:

def: P[i]=max k<i S[0...k-1]=S[i-k, ... i]

eg. abacaba
0010123




```
P-function(s)
  P[0]=0
  for i=1 to n-1
    for k=0 to i
      flag=true
      for j=0 to k-1
        if S[j]≠S[i-k-j+1]
          flag=false
      if flag=true
        P[i]=k
```



P[i] ≤ P[i-1]+1

eg. abacababc
001012320

```
P-function(s)        O(n)
  P[0]=0
  for i=1 to n-1
    k=P[i-1]
    while s[k]≠s[i] and k≥0
      k=P[k-1]
    if S[k]=S[i]
      k++
    P[i]=k
```

Problem find entry pattern P in text T

naive O(|P||T|), KMP algorithm O(|P|+|T|)

S: [P] # [T]   S=P+"#"+T   # ∉ alphabet for each Pi, Ti, Pi∉#. Ti∉#

P=P-function(S)   ~~P~~

t = abacababa

C = aba

S = aba#abacababa

P = 0010 123012323   if P[i]=|t| found.

why we need hash?

if t=aaa T=aaaaa

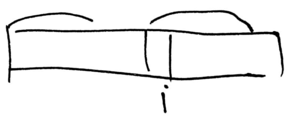
t#T = aaa#aaaaa
0120 12333

tT = aaaaaaaa ✗ the entry if P[i]≥|t|? ✗
01234567

eg. t=aba
T=aba aba
tT=001123456

still wrong

Z-function



prefix function

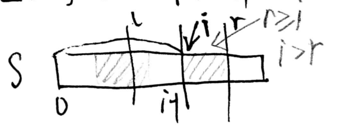
def:

$Z[i] = \max k : S[0 \dots k-1] = S[i \dots i+k-1]$

$Z[0]$ = undefined

eg. $\overline{aba}c\overline{aba}$ Z: -010301

eg. abacababab: -01030302 0



1) $r-i > Z[i-l]$ 2) $r-i < Z[i-l]$

l, r for z-block with max r

```
Z_function(s)
  l=0 r=0
  for i=1 to n-1
    z[i] = max(0, min(r-i, z[i-l]))
    while i+z[i]<n and s[i+z[i]] = s[i]
      z[i]++
    if i+z[i]>r
      l=i
      r=i+z[i]
```

t = aba

T = abacababa

t#T = aba#abacababa
-010301030301

Hashes

$h: X \to [0 \dots M-1]$

$h(S) = (S_0 \cdot p^{n-1} + S_1 \cdot p^{n-2} + \dots + S_{n-1}) \% M$ p – prime number M – bignumber $= 10^9$

$S_0, S_1 \dots S_{n-1}$ — Character codes

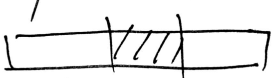
ord(s[i]) - ord('a') + 1 ← number of character in alphabet

$S_0 \dots S_{n-1} \in [0 \dots 26]$ $p = 31$ $(p \geq S_i)$.

```
~~get_hash~~ h(S)
  h=0
  for i<0 to n-1
    h=(h·p+s[i]) % M
  return h
```

$S = T \Rightarrow h(S) = h(t)$ $\Leftarrow$ not true, but little probability.

problem: entry t in text T

h(t) T 

(every substring)

```
: ht = h(t)
  for i=0 to |T|-1
    if ht = h(T[i ... i+|t|-1])      = get_hash(l,r)
      one more entry
```

power of P = 1

precalculation

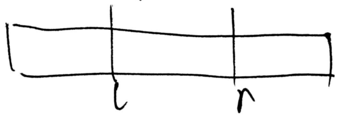
```
h[0] = s[0]
for i=1 to n
  h[i] = (h[i-1]*p + s[i]) % M
  pow_p[i] = (pow_p[i-1]·p) % M
```

$h(S[l \dots r]) = (S_l \cdot p^{r-l} + S_{l+1} p^{r-l-1} + \dots + S_r) \% M$

$h(S[l \dots r]) = h[r] - (h[l-1] \cdot p^{r-l+1}) \% M + M$;

```
get_hash(l, r)
  if l=0
    return h[r]
  return (h[r] - (h[l-1]·pow_p[r-l+1]) % M + M) % M
```

Range Query

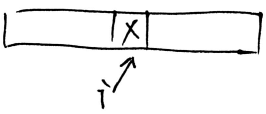
1.  range sum query (rsq) $rsq(l,r)=\sum_{i=l}^{r} a[i]$ $O(n) \to O(1)$?

$sum[i]=\sum_{j=0}^{i} a[i] \Rightarrow rsq(l,r)=sum[r]-sum[l-1]$

```
rsq(l,r)
  if l=0:
    return sum[r]
  return sum[r]-sum[l-1]
```

persudo code: ...

```
sum[0]=a[0]
for i=1 to n-1
  sum[i]=sum[i-1]+a[i]
```

2.  set(i,x)
add(i,x)

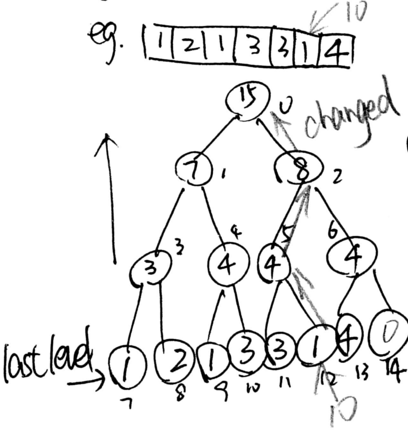
pos: 0 1 2 3 4
eg. 1 2 3 4 5   rsq(1,3) → 12
set(2,6) → 1 2 6 4 5
add(2,6) → 1 2 9 4 5

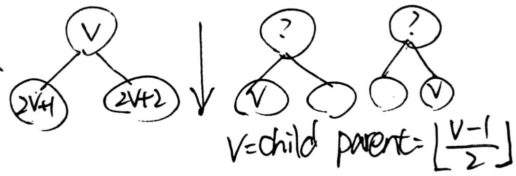
Segment Tree (rsq, add, set, rmq ...)

max/min

$a+(b+c)=(a+b)+c$ ✓ $a-(b-c) \neq (a-b)-c$ ✗ cannot use

eg. | 1 | 2 | 1 | 3 | 3 | 1 | 4 | 10





on kth level of binary tree are 2^k nodes.

h is minimal k: $2^k \geq n \Rightarrow h=O(\log n)$

number of nodes in segment Tree is O(n)

$2^0+2^1+\dots+2^h=2^{h+1}-1$

t[i] = value for i th node

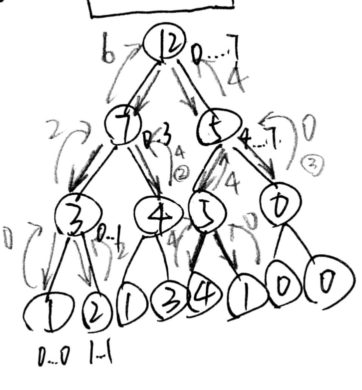
```
build(a,n)
  x=1
  while x<n        } O(log n)      <- find x => 2^x >= n
    x=x·2
  t is array of 0 with size 2x     <- initialize.   for rmq: +∞.
  for i=0 to n-1
    t[i+x-1]=a[i]                 } O(n)
  for i=x-2 to 0
    t[i]=t[2i+1]+t[2i+2]          } O(n)    <- for rmq: min(t[2i+1],t[2i+2])
```

eg. set(5,10) → 见上图

```
set(i, new-value)
  t[i+x-1]=new-value.
  cur=i+x-1
  while cur>0
O(log n) { cur=(cur-1)//2           <- integer division.
         { t[cur]=t[2cur+1]+t[2cur+2]
```

```
rsq(v,l,r,a,b)
  if l>b or r<a   //situation 3
    return 0      <- for rmq: return +∞
  if l>=a and r<=b   //situation 2
    return t[v]
  return rsq(2v+1, l, (l+r)/2, a, b) +      //situation 1
         rsq(2v+2, (l+r)/2+1, r, a, b)
         ↑
  for rmq: return min(rmq..., rmq...)
```

pos 0 1 2 3 4 5
eg. | 1 | 2 | 1 | 3 | 4 | 1 | n=6 x=8   rsq(1,4) → rsq(0,0,x-1,1,4)   (v l r a b)

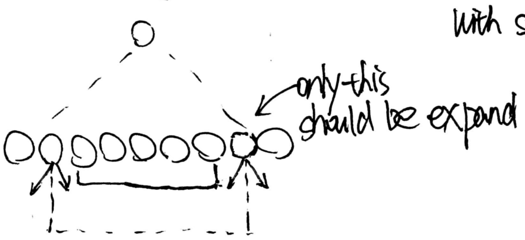


if v store sum from l,r and rsq(v,a,b)

1) $[l,r] \cap [a,b] \neq \emptyset$ and $[l,r] \not\subset [a,b]$
   return rsq(2v+1,a,b)+rsq(2v+2,a,b)
2) $[l,r] \in [a,b]$
   return t[v]
3) $[l,r] \cap [a,b] = \emptyset$
   return 0 (neural element)

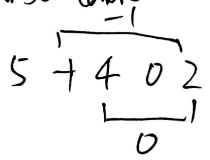
$\boxed{O(\log n)}$

why $O(\log n)$



dp: largest increasing subsequence for $O(n^2)$
with segment tree $O(n\log n)$

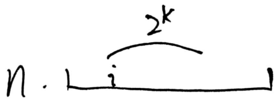
Sparse table



minimum

segment tree $O(n + q\log n)$ ↑ building

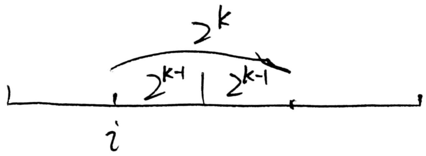
sparse table $O(n\log n + q)$ ↑ building



$0 \le k \le \log n$

$0 \le i \le n-2^k$

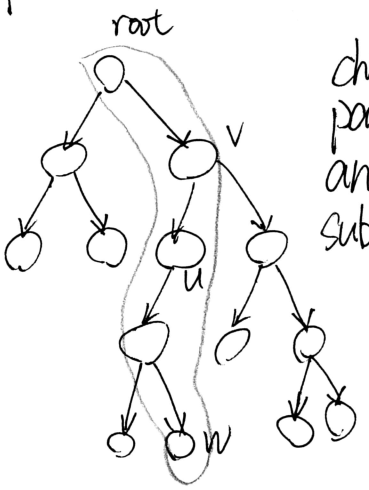
$sparse[k][i] = \min_{j=i}^{i+2^k-1} array[j]$



$sparse[0][i] = arr[i]$

$sparse[k][i] = \min(sparse[k-1][i], sparse[k-1][i+2^{k-1}])$

tree-graph



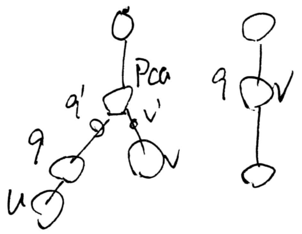
child.
parent.
ancestors
sub-tree

LCA — least common ancestor

$depth[v]$ = distance from root to V.

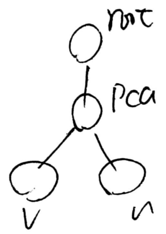
binary_jump$[v][k]$

$0 \le k \le \log n$



q is ancestor of u

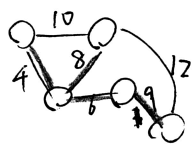
$depth[q] = depth[v]$



$depth[v] - depth[Pca] + depth[u] - depth[Pca]$

Minimum Spanning Trees & Disjoint Set Union

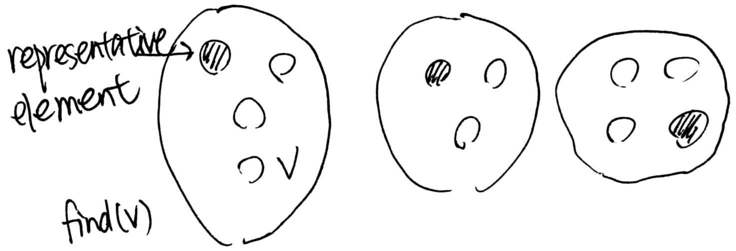
MST



Disjoint Set Union

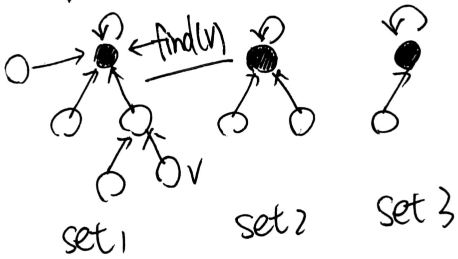
check(v,u)
merge(v,u)

V U : is it in the same node

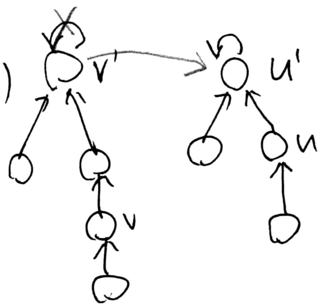


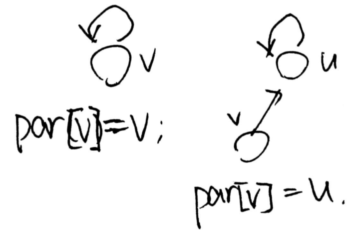
check(v, u):
  return find(v) == find(u)

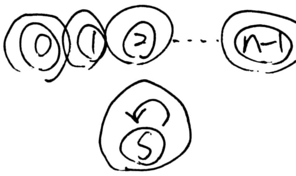
Tree-based Approach



merge(v, u)







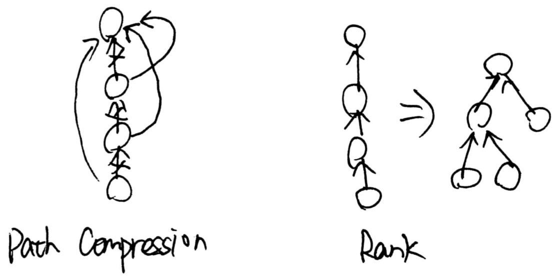
initial in different ~~grou~~ groups.

find(v): $\Theta(n)$
merge = 2× find + constant work.

1) Path Compression Heuristic

average $O(\log n)$

```
if(par[v]==v)
  return v;
int representative = find(par[v]);
par[v] = representative;
return par[v]
```

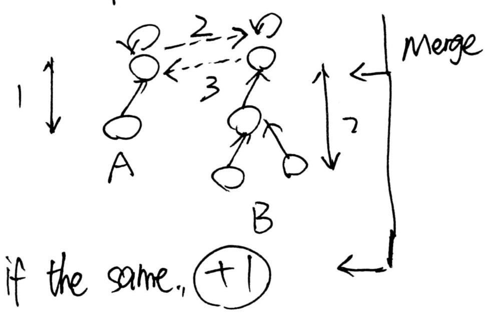


2) Rank Heuristic

```
if (rk[v] < rk[u])
    par[v] = u;
else if (rk[v] > rk[u])
    par[u] = v;
else
    par[v] = u;
    rk[u] += 1;
```

rk[v] ≤ log n

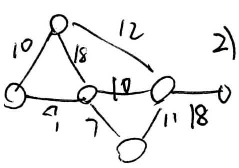
Introduction by r.



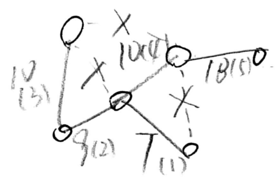
if the same, +1

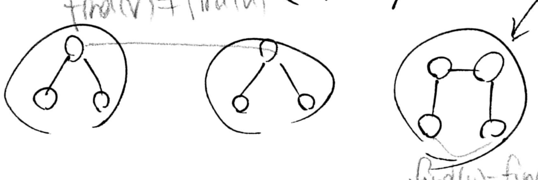
Path Compression changes find( ) Rank changes merge( ), do simutaneously ⇒ $O(\log^* n)$ or $O(\alpha(n))$

Kruskal



1) sort edges by weight
2) for (v, u, w) in Edges:
    if addition of (v, u) doesn't results in a cycle
        Add it
        Ans += w.

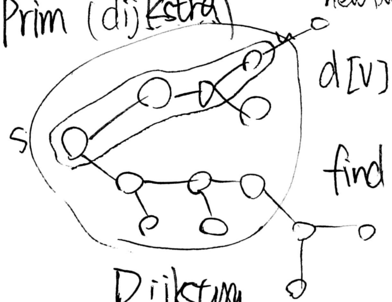




check whether it's resulted in a cycle

if in common set, find in cycle.

Prim (dijkstra) similar to

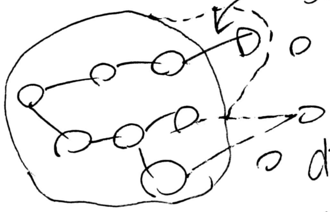


$d[V] \leftarrow \underbrace{d[u]}_{X \text{ in prim}} + w(u, V)$

find V: $d[V]$ is min

simple $O(V^2)$ ⟷ with priority queue $(E \log V)$

$E = V^2$ (worse)

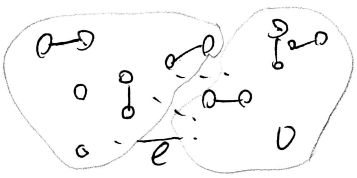


select smallest edge crossing a bubble

$d[V]$ = smallest edge into the bubble

select V outside the bubble with $d[V] \to$ min.

Safe Edge Lemma.



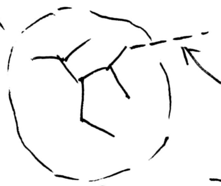
suppose we have already selected some edge correctly (a subset of some MST)

suppose we have divided/cut the graph into two parts, and no edge cross

select any smallest edge crossing the cut (eg. "e")

e can be added (still a subset of some MST)

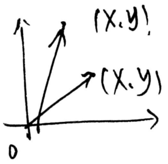
Prim

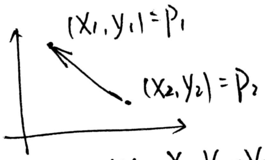


Kruskal

Basic Geometry

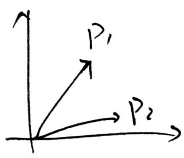
2D





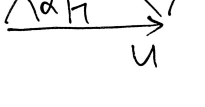
$P_1 - P_2 = (X_1 - X_2, Y_1 - Y_2)$

$P + V = (P.x + V.x,\ P.y + V.y)$



Crossproduct . $P_1 \times P_2 = |P_1||P_2| \cdot \underline{\sin(\alpha)}$ how to calculate? =

dot product $P_1 \cdot P_2$



$|u| \cdot h = |u||v| \sin \alpha$

$v = b - a$

$u = c - a$

Game Theory

2 players   1st player moves → 2nd → 1st ··· ends when player has no avaliable moves, they lose.

eg. 10 ~~points~~ coins ○○○○○○○○○○   take 1.2.3 coins once

n coins ⇒ who wins?

n	winner	
0	2	L
1	1	W
2	1	W
3	1	W
4	2	L
5	1	W
6	1	W
7	1	W
8	2	L

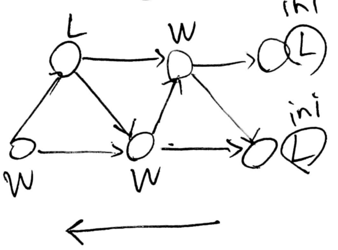
$\boxed{\begin{array}{l} \text{state} = W \Leftrightarrow \underline{\text{can}} \text{ make a move to } L \\ \text{state} = L \Leftrightarrow \underline{\text{all}} \text{ moves lead to } W \end{array}}$

state = winning
⇕
if 1st player wins

state = losing
⇕
2nd player wins.

Consider wider set of games...

Games on graphs (directed, no cycles) DAGs



1) DFS

```
vector<bool> visited, win
dfs(u):                      // returns true if u
  if visited[u]              is winning
    return win[u]
  visited[u] = true
  for u,v∈E
    if !dfs(v):
      win[u]=true
  return win[u]
```

O(V+E)

2) Dynamic Programing

a) Process vertices in reverse top ordering

```
for u
  win[u] = false
  for u→v
    if !win[v]
      win[u]=true
```

Sum of Games. G_1, G_2

Each move = 1) Choose game G_1 or G_2
2) Make a move there

eg. Nim: ① a pile with $n \geq 0$ coins
move = take any number of coins.
$n=0 \Rightarrow$ 2nd wins
$n>0 \Rightarrow$ 1st wins

② 2 piles ○○○○○ / ○○○○○ mirror tragedy (second wins)

③ 2 piles ○○○○○ / ○○○○○○○○ (the first wins)

Nim with n piles of sizes, $a_1, a_2 \cdots a_n$ is losing $\Leftrightarrow a_1 \wedge a_2 \cdots a_n = 0$

Turn out, any game G, $G \simeq \underbrace{*k}_{\text{Nim with k stones.}}$ $(\exists' k)$

$G_1 \simeq G_2 \Rightarrow$ if $\forall$game G, $G + G_1$ is W $\Leftrightarrow$ $G + G_2$ is W

k = "nimber"

"Graundy fiction" of game G

$G = \underbrace{G_1}_{\simeq *a} + \underbrace{G_2}_{\simeq *b} \Rightarrow G \simeq *(a \oplus b)$

if $k > 0 \Rightarrow G$ is winning

if $k = 0 \Rightarrow G$ is losing.

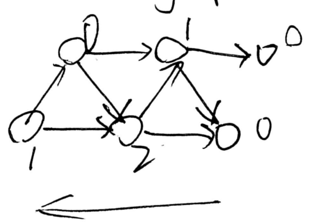
$\overset{\simeq *a_1}{G_1}, \overset{\simeq *a_2}{G_2} \cdots \overset{\simeq *a_n}{G_n} \rightarrow G \simeq *\max(a_1 \cdots a_n)$

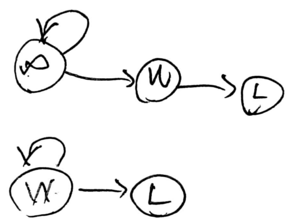
G — "choice game"   minimum excluding number.

1st move in G is:

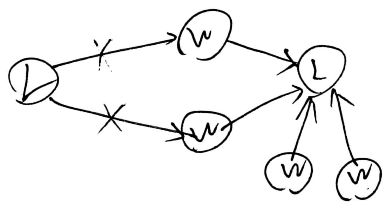
1) choose a game among $G_1, \cdots G_n$
2) continue playing in that game

Game on graphs.





Ret to grade analysis ≈ BFS



state – array of D

queue – store and processed vertics.