△ 弄清楚 edge 和 path 的区别，(u,v) 是 edge，path 可以包含多个 edge

△ Tree edge = (π[v], v), forward edge = ancester to descendant, but not tree edges
backward edge = descendant to ancester, cross edge = others (no relationship between ancester and descendant)

DFS only yields tree edge and back edge in undirected graph

△ Topological Sort = DAG

O(E) { DFS (G) — Topological sort 只是将 nodes 按顺序排一排而已
output nodes in order of decreasing finishing time

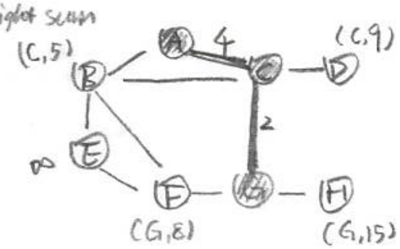
△ Strongly Connected Components

Call DFS (G) to compute finishing time f[u]
compute(G^T) — 在 G^T 上输出 Strongly connected component
Call DFS(G^T) but in the order of decreasing f[u]
output vetices of tree

△ minimum spanning tree = undirected graph, Greedy Property

Prim Algorithm: start at one vetex and grow the tree — minimize path weight sum, not number of edges in the path

for each u ∈ V[G]
    do key[u] ← ∞
        π[u] ← NIL
key[r] ← 0 — 随便选个起点开始 greedy
Q ← V[G]
while Q ≠ ∅
    do u ← Extract-MIN (Q)
        for each v ∈ Adj[u]
            do if v ∈ Q and w(u,v) < key[v]
                π[v] ← u
                key[v] ← w(u,v)



看当前 key value 和 Edge weight 大小比较

	Heap	Fibonacci Heap
V Insert	lgV	lgV
V Extract-MIN	lgV	lgV
E Decrease-Key	lgV	1
	O(ElgV)	O(E+VlgV)

Kruskal Algorithm: consider e in increasing order of weight

A ← ∅
for each v ∈ V[G]
    MAKE-SET(v)
Sort edges into nondecreasing order by weight
for (u,v) ∈ E, taken in nondecreasing order
    do if FIND-SET(u) ≠ FIND-SET(v)
        A ← A ∪ {(u,v)}
        UNION (u,v)
return A

V MAKE-SET (x) = create a one-ele set
V UNION (x,y) = merge set with x and set with y
E FIND-SET (x) = return name of set with x
After sorting, O(E $\log^*$ V)

Disjoint Sets: time per operation $\xrightarrow{\text{Proportional}}$ height of tree

Union by Rank = make a shallow tree a child of root of big tree (logV/operation)

Path Compression: every time you touch a node, make it a child of root

MAKE-SET (x)
    P[x] ← x
    rank[x] ← 0
Union (x,y)
    LINK (FIND-SET (x), FIND-SET(y))
Link (x,y)
    if rank[x] > rank[y]
        P[y] ← x
    else
        P[x] ← y
        if rank[x] == rank[y]
            rank[y] ← rank[y]+1
FIND-Set (x)
    if x ≠ P[x]
        P[x] ← FIND-SET (P[x])
    return P[x]

执行搜索才会 compression

Ackerman's Function

$A_k(j) \begin{cases} j+1 & (k=0) \\ A_{k-1}^{(j+1)}(j) & (k \geq 1) \end{cases}$

$A_0(1) = 2$
$A_1(1) = A_0^2(1) = A_0(A_0(1)) = A$

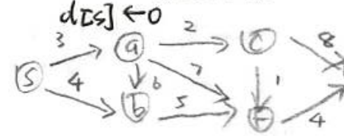
$\alpha(n) = \min\{k: A_k(1) \geq n\}$

Amortized time / operation $\alpha(V) \rightarrow \lg^* V$

△ Dijkstra's Algorithm

all edges non-negative

Initialize-Single-Source (G,s)
    for each vertex v ∈ V[G]
        do d[v] ← ∞
            π[v] ← NIL
    d[s] ← 0



Vparent	Vdist	map+heap
s→NIL	s→0	~~s→0~~
a→s	a→3	~~a→3~~
b→s	b→4	~~b→4~~
f→a c	c→5	~~c→5~~
c→a	f→6	~~f→6~~
t→a f	t→10	~~t→10~~

Dijkstra (G,w,s)
    Initialize-Single-Source (G,s)
    S ← ∅
    Q ← V[G]
    while Q ≠ ∅
        u ← Extract-MIN (Q)
        S ← S ∪ {u} — relax edges out of each vertex exactly once
        for v ∈ Adj[u] — current 与 dist 比 do Relax (u,v,w)

类似 Prim 进行 greedy

{ E decrease keys / V delete-min's
heap: O(ElogV)
fibonacci heap: O(E+VlogV)

Bellman-Ford (G,w,s) = relax edges in a shortest path in order (not necessarily consecuti

Initialize-Single-Source (G,s)
for i ← 1 to |V[G]|-1
    for each edge (u,v) ∈ E(G)
        Relax (u,v,w)
for each edge (u,v) ∈ E(G)
    if d[v] > d[u] + w(u,v) — check for negative cycle
        return False
return True

vertex	dis	parent
s	0	null
a	∞	
b	∞	
c	∞	
d	∞	
e	∞	

any order { s→a = 6, a→b = 3, c→a = 6, s→c = 2, a→d = 5, b→c = -8, c→d = 5, d→e = -1, d→b = 4 } do it |V| times (5)

Running time = O(VE)

Single-Source Shortest path (unweighted graph)

BFS

Shortest path in DAG

DAG-Shortest-path(G, w, s)

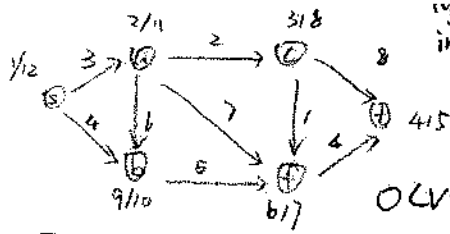
(1) topologically sort the vertices of G

INITIALIZE-SINGLE-SOURCE(G, s)

for each u taken in topological sort

each edge is relaxed once: do for each $v \in Adj[u]$

RELAX(u, v, w)



ts: S a b c f t

$O(V+E)$

Longest path in DAG

in topological order $v \in V$

$dist(v) = \max_{(u,v)\in E}\{dist(u) + w(u,v)\}$

All pairs shortest path

(1) for each vertex, run single source shortest path $= O(V^2E)$ general graph

$O(VE + V^2\log V)$ non-negative edge

(2) Floyd-Warshall (DP) $d_{ij}^{(k)} = \begin{cases} w_{ij} & k=0 \\ \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) & k \ge 1 \end{cases}$

Floyd-Warshall(W)

$n \leftarrow rows[W]$

$D^{(0)} \leftarrow W$

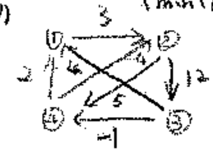
for $k \leftarrow 1$ to n

for $i \leftarrow 1$ to n

for $j \leftarrow 1$ to n

$d_{ij}^{k} \leftarrow \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$

return $D^{(n)}$



$D^0 = \begin{pmatrix} 0 & 3 & \infty & \infty \\ \infty & 0 & 12 & 5 \\ 4 & \infty & 0 & -1 \\ 2 & -4 & \infty & 0 \end{pmatrix}$ $D^1 = \begin{pmatrix} 0 & 3 & \infty & \infty \\ \infty & 0 & 12 & 5 \\ 4 & 7 & 0 & -1 \\ 2 & -4 & \infty & 0 \end{pmatrix}$ 有负环时 对角线不为0

$P^0 = \begin{pmatrix} n & 1 & n & n \\ n & n & 2 & 2 \\ 3 & n & n & 3 \\ 4 & 4 & n & n \end{pmatrix}$ $P^1 = \begin{pmatrix} n & 1 & n & n \\ n & n & 2 & 2 \\ 3 & 1 & n & 3 \\ 4 & 4 & n & n \end{pmatrix}$ $O(V^3)$

(3) w_{ij} be length of edge ij, $w_{ii} = 0$, d_{ij}^{m} be the shortest path from i to j using m/fewer edges

$\begin{cases} d_{ij}^{1} = w_{ij} \\ d_{ij}^{m} = \min\{d_{ij}^{m-1}, \min_{1\le k\le n} d_{ik}^{m-1} + w_{kj}\} \end{cases} \Rightarrow d_{ij}^{m} = \min_{1\le k\le n}\{d_{ik}^{m-1} + w_{kj}\} \,/\, O(V^4)$

With Matrix multiplication analogy = $D^1 = W$, $D^2 = D^1W = W^2$, $D^m = D^{m-1}W = W^m$ via repeated squaring $O(V^3\log V)$

(4) Johnson's Algorithm (non-dense graph) → assume strongly connected/no negative cycle

Run Bellman Ford from one arbitrary node s

use results to rewrite edges to make non-negative weights $c_p(v,w) = c(v,w) + d(v) - d(w)$

Run Dijstra for the other $n-1$ vertices

$O(VE + V(E + V\log V)) = O(VE + V^2\log V)$

MAX FLOW = min cut

Ford-Fulkerson(G, s, t)

for each edge $(u,v) \in E(G)$

$f(u,v) = 0$

while there exists a path p from s to t in residual network G_f

$c_f(p) = \min\{c_f(u,v) : (u,v) \text{ is in } p\}$

for each (u,v) in p

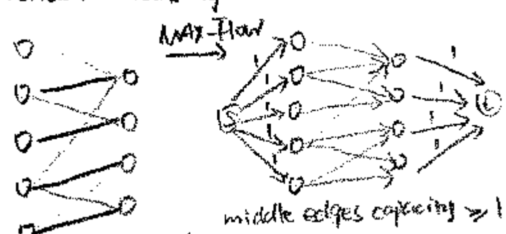
if $(u,v) \in E$

$f(u,v) = f(u,v) + c_f(p)$

else $f(v,u) = f(v,u) - c_f(p)$

1 iteration takes $O(V+E)$, sends at least one unit flow

total time $= O(f \cdot E)$



Maximum Bipartite Matching

M为边集E的子集，若M中任意两条边不依附同一顶点，则称M为一个matching

Maximum matching 所选M中边数量最多

Linear programming

(1) Max- s-t Flow = $f(u,v)$, $(u,v) \in E$

Maximize $\sum_{(s,v)\in E} f(s,v) - \sum_{(v,s)\in E} f(v,s)$

Subject to: $f(u,v) \ge 0$ non-negativity

$f(u,v) \le c(u,v)$ capacity constraints

flow conservation constraints $\sum_{(u,v)\in E} f(u,v) - \sum_{(v,u)\in E} f(v,u) = 0, \forall u \in V - \{s,t\}$

(2) Minimum Cost flow

a way of minimizing the cost required to deliver max amount of flow possible in the network

$\min \sum_{(u,v)\in E} cost(u,v) f(u,v)$

s.t. $\sum_{(u,v)\in E} f(u,v) - \sum_{(v,u)} f(v,u) = 0, \forall u \in N - \{s,t\}$

$\sum_{(s,v)\in E} f(s,v) - \sum_{(v,s)\in E} f(v,s) = demand$

$f(u,v) \le c(u,v)$ flow in the network

$f(u,v) \ge 0$

min cost flow with all capacity = 1, demand = 1 ⟹ Shortest s-t path

Solutions of LP: infeasible, unbounded, finite optimum

P = solved in polynomial time ($O(n^k)$ for some constant k)

NP = verified in polynomial time

$Y \le X$ = polynomial f maps inputs to Y to inputs to X → complexity

inputs to y return "Yes" iff inputs to x returns "Yes" (reduce Y to X)

$Y \le X \rightarrow X \in P \Rightarrow Y \in P \,/\, Y \notin P \Rightarrow X \notin P$

Y is easy, $X \le Y \rightarrow X$ is easy; 若 Y is hard, $Y \le X \rightarrow X$ is hard

To show X is NP-Complete:

(1) $X \in NP$

(2) find a known NP-complete problem Z

(3) describe f which maps input to Z to $f(x)$ to X

(4) Show Z with inputs x return "yes" iff X with inputs $f(x)$ returns "Yes"

(5) Show f is polynomial time

SAT = NP-complete

3SAT = SAT with exactly 3 literals per clause

$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_4 \vee \bar{x}_5)$

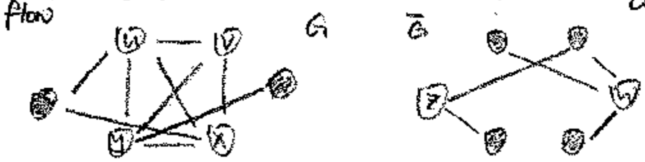
$\phi = x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$ to 3SAT

$\phi' = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge (\bar{y}_2 \vee x_4 \vee x_5)$

NP-complete:
Clique = set of k vertices with all $\binom{k}{2}$ edges between them
Vertex Cover = vertices that cover edges

G has a k-clique iff G' has a vertex cover of size $|V| - k$

(for G, S is an independent set iff the set $V - S$ is a vertex cover)



clique $V' = \{u, v, x, y\}$ vertex cover $\{w, z\}$

Relax(u, v, w)

if $d[v] > d[u] + w(u,v)$

$d[v] \leftarrow d[u] + w(u,v)$

$\pi[v] \leftarrow u$

Big-O= $0 \le f(n) \le c\cdot g(n)\ \forall n \ge n_0$

Big-Ω: $0 \le c\cdot g(n) \le f(n)\ \forall n \ge n_0$

Big-Θ = $f(n) = O(g(n))$ & $f(n) = \Omega(g(n))$

Little-O= $0 \le f(n) < c\cdot g(n)\ \forall n \ge n_0, \forall c>0$

$\{ \lim_{n\to\infty} f(n)/g(n) = 0$

Little-W= $0 \le c\,g(n) < f(n)\ \forall n \ge n_0, \forall c>0$

$\{ \lim_{n\to\infty} f(n)/g(n) = \infty$

<u>, logn, n, nlogn, n^2, n^3, 2^n, n!</u>

Arithmetic series $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$

Geometric Series $\sum_{i=0}^{\infty} a^i = 1/(1-a) \quad 0<a<1$

$a + ar + ar^2 + \dots + ar^{n-1} = \sum_{k=0}^{n-1} ar^k = a\left(\frac{1-r^n}{1-r}\right)$

Harmonic Series $\sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1) = \Theta(\ln n) = O(\log n)$

MergeSort $O(n\log n)$ $T(n)=2T(n/2)+O(n)$ ⇒ B.S: $O(\log n)$ $T(n) = T(n/2)+O(1)$

QuickSort $O(n\log n)$ best case / $O(n^2)$ worst

InsertSort $O(n^2)$ best case $O(n)$ $T(n) = T(n-1)+O(n)$

HeapSort $O(n\log n)$

Master theorem = $T(n) = aT(n/b) + f(n)$

$n^{\log_b a} > f(n) \to \Theta(n^{\log_b a})$ $f(n)=O(n^{\log_b a-\varepsilon})$, for some $\varepsilon>0$

$n^{\log_b a} = f(n) \to \Theta(n^{\log_b a}\cdot \lg n)$ $f(n)=\Theta(n^{\log_b a})$

$n^{\log_b a} < f(n)$ & $af(n/b) \le c\cdot f(n) \to \Theta(f(n))$, $c<1$; $f(n) = \Omega(n^{\log_b a+\varepsilon})$, for some $\varepsilon>0$

Proof by induction = format

claim = $T(n) \le cn^2 - dn$ for some $c,d>0$

proof = ...

conclusion = ...

$T(n) = 2T(\sqrt{n}) + \log n$

$m = \log n$, $T(2^m) = 2T(2^{m/2}) + m$

$S(m) = 2S(m/2) + m$

$S(m) = \Theta(m\log m)$, $T(n) = \Theta(\log n \log\log n)$

$\log(n!) = \Theta(n\log n)$

△ binary search 1/3, 2/3, worst case

$T(n) = T(\frac{2}{3}n) + 1 \Rightarrow T(n) = \log_{\frac{3}{2}} n$

△ MergeSort 1/3, 2/3, $T(n) = T(n/3) + T(2n/3) + n$

$T(n) = \Theta(n\log n)$

HEAPSORT (A)

1. BUILD-MAX-HEAP (A)
2. for i = A.length downto 2
3. exchange A[1] with A[i]
4. A.heapsize = A.heapsize − 1
5. MAX-HEAPIFY (A, 1)

(lines 3–5: Extract MAX)

BUILD-MAX-HEAP (A)

1. A.heapsize = A.length
2. for i = ⌊A.length/2⌋ downto 1
3. MAX-HEAPIFY (A, i)

- Let Xi be the indicator random variable associated with the event in which ith flip comes up heads
- $X_i = I\{$the ith flip results in the event H$\}$
- Let X be the random variable denoting total number of heads in the n coin flips
- $X = \sum_{i=1}^{n} X_i$ $E[X_i] = 1 \times \frac{1}{2} + 0$

$E[X] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} 1/2 = n/2$

△ $\frac{n!}{(n-k)!}$ possible <u>k-permutation of n elements</u>

△ $\binom{n}{i}$ Select i from n, $\frac{n!}{i!}$

worst $T(n) = T(n-1) + O(n)$

QuickSort (A, p, r) $T(n) = 2T(n/2) + O(n)$ $= O(n\log n)$

1. if p < r
2. then q ← PARTITION (A, p, r)
3. QuickSort (A, p, q−1)
4. QuickSort (A, q+1, r)

☆ PARTITION (A, p, r) $O(n)$

1. y ← RANDOM(p, r)
2. Exchange A[y] and A[r]
3. x ← A[r]
4. i ← p−1
5. for j ← p to r−1
6. do if A[j] ≤ x
7. then x ← x+1

9. for ... Exchange A[i+1] ↔ A[r]
10. return i+1

CountingSort: $D = \{1, \dots, k\}$

running time $\underline{\Theta(n+k)}$

BucketSort: bucket $Q[i]$, for $i = 1, \dots, k$

Record $A[j]$, for $j = 1, \dots, n$

running time $\begin{cases} \Theta(k + n \cdot |record|) \text{ if copy whole records} \\ \Theta(k+n) \text{ if only indices of records} \end{cases}$

Radix Sort: d-digits number, each digit takes k possible values.

or d-vectors, where ith Component takes k_i values

running time: $\underline{\Theta(d(n+k))}$

constants: π^{2019}

doubly logarithmic: $\log\log n$

logarithmic: $\log n + \log\log n$, $\log(n^{0.19})$

super-logarithmic but sub-polynomial: $2^{\sqrt{\log n}}$

Square-root: $\log(2^{\sqrt{n}})$

nlogn: $n\log n$, $\log(n!)$ $\begin{cases} \log(n!) \le \log(n^n) \le n\log n \\ \log(n!) \ge \log((n/2)^{n/2}) \ge (n/2)\log(n/2) \end{cases}$

polynomial: $n^{3+\sin n}$

polynomial with higher degree: $\binom{n}{n-19} = \binom{n}{19} = \Theta(n^{19})$

super-polynomial but subexponential: $n^{\log\log n}$, $(\log n)^{\log n}$ equal

mild exponential dependence: $3^{\sqrt{n}}$

exponential with power 2: 2^n, $\underline{2^{2\log n + 19} = 2^{n+19} = 2^{19} \cdot 2^n}$

exponential with higher base: $\underline{2^{2(\log n + 19)} = 2^{2^{19} n}}$

$\sum_{n=1}^{\infty} 1/n^2 = \frac{\pi^2}{6}$

heapsort Application:

Top k largest: $O(n + k\log n)$

TopK online Stream $(n \gg k)$: use minheap $O(n \cdot \cancel{k}\log k)$

Hoffman Code (C)
```
  n ← |C|
  Q ← C
  for i ← 1 to n-1
        new node z
        left(z) ← Extract-MIN(Q) x
        right(z) ← Extract-MAX(Q) y
        f(z) ← f(x) + f(y)
        Insert(Q, z)
  return Extract-MIN(Q)
```

▲ Aggregate Analysis

$\begin{cases} m(i) \text{ number of pops done in ith multipop} \\ P \text{ the number of push done overall} \end{cases}$

$\sum_i m(i) \le P$

time = pushes + time for all multipops

$= P + \sum_i m(i) \le P + P \Rightarrow P \le 2n$

证明对所有n,一个n个操作序列总时间为T(n),则平均为O(1)

▲ Banker's method

$\forall l \; \sum_{i=1}^{l} \hat{c}_i \ge \sum_{i=1}^{l} c_i$

each operation has real cost c_i and amortized cost $\hat{c}_i$

对每个操作收费为amortized cost,当个操作的amortized cost超过实际代价,差为credit,用于补偿amortized cost小于实际的操作.

▲ potential function

$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

define the right potential function.