README

▣ 笔记总结自 AI2615 Spring 2024 上课内容，主要为笔者以自己的理解和视角描述了课上提到的所有算法和分析过程

▣ 授课内容为：分治，图论，贪心，动态规划，网络流，线性规划，NP 问题

▣ 其中偶尔记录了考纲范围外的内容，比如 Cook-Levin 定理，(F)PTAS，Fixed-Parameter Tractable 等。因为笔记的初衷并不完全为了 cheating paper，更多为了记录讲课思路本身

Intro

1. Halt Problem : Halt(TM, y) = { 1. if Turing Machine (y) halt ; 0, not halt. }

(证明不是所有 decision Problems 都能解 — 反例 Halt problem)

因为可以 def B(x):
    if Halt(x,x)=0: return 1
    else: for(;;);

B(B) Halt 的分支要求 B(B) 不 Halt

则 B(B) → 判断 Halt(B,B) —=0的分支要求→ B(B) not halt

2. $O(g(n))$. $\exists C, n_0, \forall n>n_0$. $T(n) \le C\cdot g(n)$

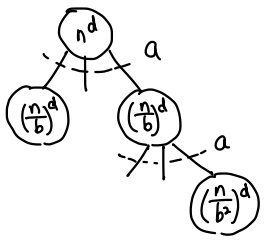
$\Omega(g(n))$. $\exists C, n_0, \forall n>n_0$. $T(n) \ge C\cdot\Omega(n)$

$\Theta(g(n))$. $T(n) = T(g(n)) = \Omega(g(n))$

Divide

1. Master Therom

$T(n) = a\cdot T(\frac{n}{b}) + c\cdot n^d \log^w n$



$$\Rightarrow T(n) = \begin{cases} O(n^d \log^w n), & a<b^d \\ O(n^{\log_b a}), & a>b^d \\ O(n^d \log^{w+1} n), & a=b^d \end{cases}$$

2. Karatsuba

要计算 ac, ad, bc, bd 四项. 分治后还是 n^2

但 ad+bc 整体等于 (a+b)·(c+d) − ac − bd

$x\cdot y = (a\cdot 10^{\frac{n}{2}} + b)\cdot(c\cdot 10^{\frac{n}{2}} + d)$

$= ac\cdot 10^n + (ad+bc)\cdot 10^{\frac{n}{2}} + bd$

$= ac\cdot 10^n + [(a+b)(c+d) - ac - bd]\cdot 10^{\frac{n}{2}} + bd$ 三次乘法

$T(n) = 3T(\frac{n}{2}) + c\cdot n \Rightarrow T(n) = O(n^{\log_2 3})$

若位数不是 2^k, 则左侧补零

3. ① Divide — into Small Subproblems

② Recurse — solve subproblems reursively

③ Combine — combine outputs of subproblems

④ Basic Solver

4. Merge Sort

$T(n) = 2T(\frac{n}{2}) + c\cdot n$ 分成两半分别排序, 线性时间合并

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

基于比较的算法至少 $O(n\log n)$. 因为每次比较 3 种结果 <, =, >

比较 k(n) 次就能分辨 $3^{k(n)}$ 情况, 但总排序情况 $P_n^n = n!$

$\therefore 3^{k(n)} \ge n!$ $k(n) \ge \log_3(n!)$. $k(n) = \Omega(n\log n)$

5. Select k-th Smallest

① 选 K 次最小的 $O(nk)$ ② 先排序 $O(n\log n)$

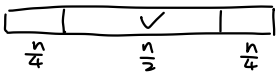
③ 先选一个 pivot 将数组分成 3 部分 {x<v} {x=v} {x>v} $O(n)$

```
if k ≤ |L| : return Select(L, k)
elif |L|<k≤|R| : return v
elif |L|+|M| < k : return Select(R, k)
```

问题在于. 若每次 pivot 都很靠近边缘. 则只减 1 长度 1

$O(n)+O(n-1)+\cdots = O(n^2)$

但平均分析: Choose pivot randomly (not arbitary)



若 pivot 在中间 $[\frac{n}{4}, \frac{3n}{4}]$ 处 则分成的三段中最差时最长 $\frac{3n}{4}$

$\therefore T(n) \to T(\frac{3n}{4}) + C\cdot n$

若每次都能选在 V 区, 则 $T(n) = T(\frac{3n}{4}) + C\cdot n \Rightarrow O(n)$

但实际上第几次选在 V 区是二项分布. $p = \frac{1}{2}$

则期望 $\frac{1}{p} = 2$ 次选到处于 V 区. 即期望 2Cn 就到 V 区

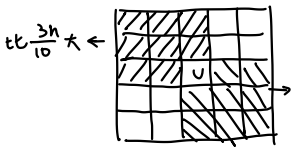
$\therefore E[T(n)] = E[T(\text{选到V区}) + T(\frac{3n}{4})] = C'n + E[T(\frac{3n}{4})]$

$\Rightarrow E(T(n)) = O(n)$.

④ 平衡选 pivot 时间和好处 —— 若排序可选中. 但慢 / 若 random 选很快. 但可能差

选择选 pivot:

- 比如分成 5 个一组: $O(n)$
- 每组中找 middle : $O(n)$ 因为每组就 5 个数. 直接硬比
- medians 的 middle 作 v: $T(\frac{n}{5})$ 即 Select $(M, \frac{|M|}{2})$



故 pivot 一定在 $[\frac{3n}{10}, \frac{6n}{10}]$ 区间内. 分成的三段最差最长 $\frac{7n}{10}$

$\Rightarrow T(n) = T(\text{选pivot}) + T(\frac{7n}{10})$

$= T(\frac{n}{5}) + C\cdot n + T(\frac{7n}{10})$

归纳法证明: 猜 $T(n) \le B\cdot n$

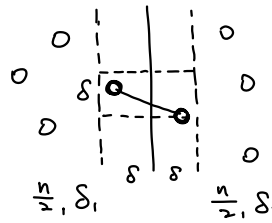
$T(1) = 1 \le B\cdot 1$ 取 $B = 10C$

$T(n) = T(0.2n) + T(0.7n) + Cn \le 0.9Bn + Cn = Bn$

$\Rightarrow T(n) \le B(n)$ 归纳得证.

错误. 归纳不能用 $O(n)$ 记号, 必须写出 C·n 常数是多少. 因为每次常数都扩大. 归纳 n 次后不再是常数

6. Closest Pair



分成两部分, 各找自己的最短距 δ:

$\delta = \min\{\delta_1, \delta_2\}$

① 若下一步遍历中间 $[-\delta, \delta]$ 区域最坏情况 n 全在里面 $O(\frac{n}{2}\times\frac{n}{2})$

② 但当也给 y bound $[-\delta, \delta]$. 后. 对内点的数量就 bound 了 ≤ 8

$\Rightarrow$ 至多比较 7 次. (7 不重要)

$\Rightarrow T(\text{遍历中间}) = T(\text{排序}y) + \text{中间点数}\times 7 = O(n\log n + 7n)$

$\therefore T(n) = 2T(\frac{n}{2}) + O(n\log n) \Rightarrow T(n) = O(n\log^2 n)$

③ 但实际 排序 x 与 排序 y 的 $O(n\log n)$ 可分别从递归中消掉

↓ 上来先排好. 用一次 $O(n\log n)$

↓ 除法先排好 (每次要 S' 的顺序就遍历总 y 序. 然后检查 x 坐标是否在范围)

还可以用下层传上来的两个排好的 S_1', S_2'.

因此 $T(n) = 2T(\frac{n}{2}) + O(n) = O(n\log n)$ 再加预先 $O(n\log n)$

7. 多项式乘法 — 应用 FFT

DFT 矩阵 $F_{ij} = \omega^{ij}$, $\omega = e^{i\frac{2\pi}{n}}$ ← 对于 n 的多项式.

多项式 $A(x) = a_0 + a_1 x + \cdots + a_n x^n$, 可以只记系数 $\vec{a} = \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix}$

FFT 算法 — 在 $O(n\log n)$ 内计算 $F\cdot\vec{a}$, 即 A 在 n 个单位根上的值

① 算 A 在这些点上的值 $F\cdot\vec{a} = \hat{a}$ ② $F\cdot\vec{b} = \hat{b}$ ③ $\hat{c}_j = \hat{a}_j \times \hat{b}_j$

④ 为得到 $\vec{c} = F^{-1}\hat{c} = (\frac{1}{N}F)\cdot\hat{c} = \frac{1}{N}\overline{(F\cdot\hat{c})}$

①②④ 三次 $O(n\log n)$. ③ $O(n)$. $\Rightarrow$ 总时间 $O(n\log n)$

8. FFT. 具体如何算在 n 个单位根上插值. $p(\omega^0), p(\omega^1) \cdots p(\omega^{n-1})$

注意 $p(z) = p_0 + p_1 z^1 + p_2 z^2 + p_3 z^3 + p_4 (z^2)^2 + p_5 z^5 + p_6 (z^2)^3 + \cdots$

$= P_{even}(z^2) + z \cdot P_{odd}(z^2)$ ，其中 P_{even} P_{odd} 是取一半系数的多项式

⇒ n 个点需要算 P_{even} on $\omega^0, \omega^2, \omega^4, \cdots \omega^{2n-2}$

P_{odd} on $\omega^0, \omega^2, \omega^4, \cdots \omega^{2n-2}$

返仍是 $2\times\frac{n}{2}$ 个插值. 关键①：$\omega^0 \sim \omega^{n-1}$ 平方之后是两圈重合

即变成 (ω^2) 为单位根绕2圈. 只要算 $2\times\frac{n}{2}$ 个插值. 再花 $O(n)$ 合并

$T(n) = 2T(\frac{n}{2}) + O(n)$ ⇒ $T(n) = O(n\log n)$

↳ 关键②：$\omega^0, \omega^2, \omega^4$ 再平方后. 是以 ω^4 为单位根.

即 divide 可以一直做到底.

Graph

一、基础

1. $G(V, E)$. $0 \le |E| \le |V|(|V|-1) = O(|V|^2)$
2. 临接矩阵 space $O(|V|^2)$

   临接表 space $O(|V|+|E|)$
3.
```
def explore(v):
    marked[v] ← true
    for each (u,v) ∈ E
        if marked[u] == false: explore(u)
```

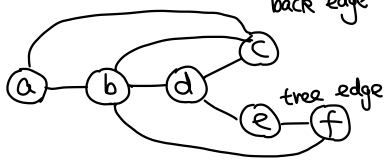
一次独立 explore 可以找全一个连通分量. 但图不一定连通.

∴ DFS 不停新 explore:

```
def DFS(G):                                   O(|V|+|E|)
    for each v ∈ V:
        if marked[v] == false : explore(v)
```

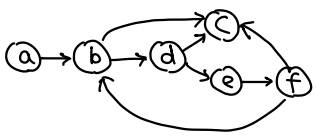
进入几次这里↑. 无向图就有几个连通分量

4. 无向图的 DFS tree



有 back edge ⇔ 有环

有向图 的 DFS tree



Tree
Forward
Back
Cross
} edge

5. topological order

有向无环 DAG ⇔ 存在 topological order

"⇐" 用反证 若有环必无序

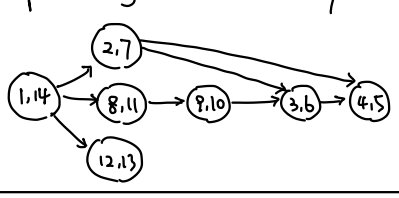
"⇒" 通过对任意 DAG 构造拓扑序来证

① DAG 必有 tail ② delete tail ⇒ new DAG ③ delete new tail…

然而每次找 tail 太重复了. ⇒ $O(|V|^2)$

Improve by DFS：记录 start/finish

DAG 没有 back edge



实现：每结束一个就记录

按倒序输出

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

```
time ← 0
def explore(v):
    start[v] ← time ; time++
    marked[v] = true
    for each (u,v) ∈ E:
        if marked[u] == false: explore(u)
    finish[v] ← time ; time++
```

- Earliest finish time is tail
- If finish[v] > finish[u], no (u→v)

  proof：枚举所有三种可能的边类型.(无back)

① 若 u→v 是 tree edge，说明 v 在 u 子树里. DFS 必先退出.

② 若 forward edge，也说明 v 在 u 子树里.

△③ 若 u→v 是 cross edge. 存在的唯一可能是 DFS 先走到 v，否则 cross 就变成 tree edge 了. 分支就合并了.

⇒ 所以 v 必定完全完成后才进行 u. finish[u] < start[v]

重要思考点：若有向图 DFS 有 cross edge，必然先走的终点那支！

- finish time 的降序就是拓扑序

6. Strong/weak Connect Component

↓ 改成无向后的 CC

$\forall u, v \in SCC_i$，u, v 能互相到达，且是 maximal CC

每次都要从 "tail SCC" 中的某个点起始 DFS. 否则跨 SCC.

最晚结束的点(finish[u]最大) 一定在 head SCC

(Intuitively，一个点结束，意味所有它能到的点都探索完了. 因此最晚结束应在全体的 head SCC 中)

证明：反证若不在 head SCC 中. 记 u 在 SCC1，head 是 SCC2

则 SCC2 中有到 SCC1 的 path. 即 $\exists v \in SCC2, x \in SCC1$，

有路径 $v \to x \to \cdots \to u$

矛盾点：由于 u 结束最晚. 所以 u 必不在 v 的子树中，

不仅如此，v 还必须在 u 的子树中. 否则还是 u 先结束.

⇒ 有路径 $u \to \cdots \to v$. 存在环 ↯.

Basic 实现：每次 DFS 新的反图，逐次去点建新图：$O(n(n+m)) = O(nm)$

Super 实现：先 DFS 一次存所有点的 finish time，

按降序依次 DFS，总共是一次完整 DFS ⇒ 共2次 DFS $O(n+m)$

依赖定理：(HW2.1证明)

If SCC1 reach SCC2，则 SCC1 中最大 finish > SCC2 中最大 finish

[补上页分析 pivot 的平均表现]：可以等价于认为：

以 $\frac{1}{2}$ 概率分到 lucky. 变成 $T(\frac{3n}{4}) + C\cdot n$.

另 $\frac{1}{2}$ 的情况可认为什么都没做 $T(n)$.

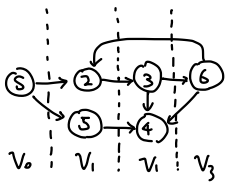
$\therefore T(n) = \frac{1}{2}\cdot\left[T(\frac{3n}{4}) + Cn\right] + \frac{1}{2}T(n) \Rightarrow T(n) = T(\frac{3n}{4}) + C\cdot n$

↳ 但实际不是什么都没做，而是按 unlucky 分的段做 $T(l)$，$\frac{3n}{4} < l \le n$

我们放缩成了 $T(n)$. 若其它题是复杂函数不单调. 不能这样放缩

二、Path

1. BFS tree



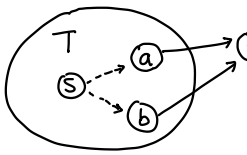
- hard to distinguish cross & back edge
- finish time 没用

```
def BFS(G,s):
    i ← 0
    V0 ← {s}  第0层的点
    while Vi is not empty:
        for each u∈Vi:
            for each (u,v) ∈ E:
                if marked[v] = false:
                    marked[v] = true
                    add v to V_{i+1}
                    pre[v] = u   ← 记录一下是谁访问的v. ⇒ shortest path
        i++
```

2. Short Path tree — 从根s到任意u的树也是最短路

• 存在 SPT → 给出一种对任意图的构造方式：Dijkstra.

归纳证明：从 $\{s\}$ 开始逐渐拓展



$|T|=1$ 成立.

设已有部分 T，即 $\forall u\in T.\ dist_T(s,u)=dist(s,u)$

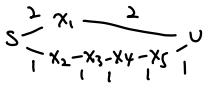
如果下一个添加的点是到 T 最短的（不是 $d(u,v)$，是 $dist_T(v)$）

即 $dist_T(s,v) = \min_{u\in T}\{dist_T(u)+d(u,v)\}$ 是最短的 v

此时若想经过其它非 T 中点到达 v，第一步 $d(u,x)$ 就 $> d(u,v)$

若真的 $\{T\}\to x\to v$ 更短，则这轮应该先添加 x

∴ 经过/不经过 T 以外的点，添加这个 v 都最优，仍 SPT



$\{s,x_2\}\to\{s,x_2,x_3\}\to\{s,x_2,x_3,x_1\}$

不是添加"新边最短的"

而是 $dist_T(v)$ 最短的 v，即只用 T 最短的

```
def Dijkstra(G,s):   单源  正边权
    1. Initialize
        T = {s}.
        tdist[s] = 0,  tdist[v] = ∞ for other vertices
        tdist[v] = d(s,v)  for (s,v) ∈ E   ← 这也是update
    2. Explore
|V| rounds   find v∉T with min tdist[v]
        T ← T + {v}
    3. Update tdist[v]   ↑ 为了维护 tdist[v] 时刻是从当前{T}能抵达v的最短路
        tdist[u] = min{ tdist[u],  tdist[v] + d(u,v) }
|E| rounds                                  for (u,v) ∈ E
        ↳ 只用新加的点v与原来比较来更新
```

因为每条边仅会产生2次update

$$T(总) = |V|\cdot T(Pop\ min) + |E|\cdot T(Update)$$

↓ find min + 删，以后不找了　　↓ 某个结点变小了，重新维护

	pop min	Ins	Update	merge
d-array	$O(d\log_d n)$	$O(\log_d n)$	$O(\log_d n)$	$O(n)$
Binomial	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n)$
Fibonacci	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$

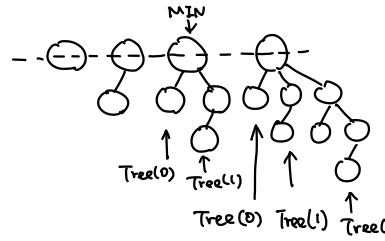
① 若用数组：$T(n)=O(|V||V|+|E|)=O(n^2)$

② 若2叉堆：$T(n)=O((|V|+|E|)\log|V|) \xrightarrow{稠密} O(n^2\log n)$

③ Fibonacci：$T(n)=O(|V|\log|V|+|E|\times 1)=O(m+n\log n)$

3. Fibonacci Heap — 均摊分析 △ Amortized Analysis

基本想法：为了避免这些树退化成数组，bound 度数.



① 从低向高每个度数的树最多有一个.

② 不仅如此，child也要满足不同度数最多一个，度为k的根要有度0~k-1的子树.

⇒ **组成一棵度为k的树至少需要 2^k 个点**

（因为 $d(k)=\sum_{i=0}^{k-1} d(i)+1=2^k$）

n个点能组成的最大度数的树为 $\log n$

因为 popmin 的更新最小指针需 $O(t^-+D)$ { t^-：根数量；D：最大度数

要分别bound，由上面性质①②. $D\le\log n$.

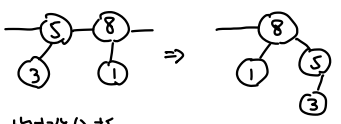
③ Cascating Cut Rule：每棵树至多可失去一个child.

否则将所有第2次失去儿子的点都设为根

但③会造成很多根，所以用 Merge 操作来 bound t^-

④ Merge Rule：每次新生成很多根后，将相同度数合并.

可 inductive 证明新树仍满足 { 一个度数子树至多一个；可失去一个儿子 } ⇒ $D\le\log n$



- 均摊分析　→ $\hat{C}=O(k)+\delta(-k)=O(1)$

比如某种 Stack：push-one，pop_all.

pop_all 花费 $O(k)$，但有 k 个元素前提一定有 k 次 $O(1)$ push ⇒均摊

- Potential Function　（随意chosen常数）

$\Phi_0=0$. represent states. $\hat{C}=C+\delta\cdot\Delta\Phi$

$\Sigma\hat{C}=\Sigma C+\delta\cdot\Phi$，（能均摊要求 $\Phi\ge 0$ 即 $\Sigma\hat{C}\ge\Sigma C$）

⇒均摊 $O(\frac{\Sigma\hat{C}}{N})$ 操作次数　　↓ 即必须向差补偿.

- Update：$O(m')$ → 需合并次数

令 $\Phi=t+2m$

↓ 因为 m 会多造成 cascating cut 以及潜在多一次 merge

#CC cascating cuts，#CC marks，#CC roots

$C=O(\#CC+1)$

$\Delta t=\#CC+1$

$\Delta m=-\#CC+1$

$\hat{C}=O(\#CC+1)+\delta(-\#CC+3)=O(1)$

- Pop Min：$C=O(t^-+D)$，$\hat{C}=O(t^-+D)+\delta\cdot\Delta t$

$\Rightarrow \hat{C}=O(t^-+2D)+\delta(D-t^-)=O(D)=O(\log n)$

三、Negative Weight

Dijkstra 在负边权不工作. 因为虽然 T 到 U 最短. 但可能 $dist_T(x) + c(x,u) \ll dist_T(x) < dist_T(u)$

本质. Dijkstra 每边用过后就不再检查. Bellman-Ford 仍持续更新

def bellman_ford(G,s): 单源 负边权

dist[s]=0, dist[x]=∞ for other x∈V

while dist is updated: 每次更新所有边.

for each (u,v)∈E

dist[v] = min{dist[v], dist[u]+d(u,v)}

- 正确性两步 ① 停止时是 SPT ② 一定会停止

Lemma1: k 轮更新后. dist(v) 是至多 k 长路的最短路长

Inductive: 若所有 k-1 都最短. (s) ⇢ (u_{k-1}) → (v)

算法枚举了所有能到 v 的 u_{k-1}: $\min_{(u_{k-1},v)\in E}\{d_{k-1}(u_{k-1}) + w(u_{k-1},v)\}$

Lemma2: |V| 长的路一定比 |V|-1 长的路优

因为 |V| 有环. 则不走这个环一定更优. ⇒ 必在 |V| 轮停止

⇒ 若算法在 |V| 轮还在更新可判断有负环

Greedy

1. 任务完成. n 个任务 with ddl: d[i] and time cost S[i].

定义目标: 完成所有 Greedy 策略: 先做 ddl 最早的.

最优性: 若 input 能完成, 则 Greedy 必能完成

证其逆否: 若 Greedy 不能完成, 则必无法完成

↓

$\exists i$. 使 $\sum_{j\le i} S_i \ge d_i$ 其中前 i 个任务满足 $\forall j<i.\ d_j < d_i$

⇒ 无法在前 d_i 时间内完成 1~i 号任务, 必须至少挪出一个到后面

但这 i 个任务 ddl 都 $< d_i$. 则挪出哪个都失败.

2. Minimum Spanning Tree → 边无环且包含所有点 → |V|-1 条 → 树.

Prim: 和 Dijkstra 一模一样更新每个点到 T 的距离.

每次向 T 添加一个距离最短的点 ⇒ |E| round Update, |V| Popmin

⇒ Use Fibonacci heap O(|E|+|V|log|V|)

Kruskal: 将边排序, 按升序依次选, 若成环则不选

① Sort Edges O(ElogE) (忽略这个瓶颈, 没 ordered)

② for each (u,v)∈E in 升序: ← 2|E| round check.

if group(u) != group(v):

choose (u,v)

union (group(u), group(v)) ← total |V| round.

Use Union-Find Set.

Union: O(1); Find (查 group) 均摊来说 < SamGroupC 共: $n\log^* n$ / CrossGroupC 共: $m\log^* n$

∴ 总时间 $O((m+n)\log^* n)$

均摊原理: rank 小的点很多. 大的点少 ⇒ 将小点分散分组.

[1,2] [2,2^2] [2^2,2^{2^2}] … 兼顾 bound 了组内/间的 Find.

Greedy 证明思路 { 反证: 若交换任意两元素. 结果不会变好 / 归纳: 设最优 F_n, 推 F_{n+1} 一定用该策略 } → 或 Greedy 不会 ruin F_n 的最优

3. 选择任务 (目标: 总完成数量最多)

贪心: 从它始总做当前能做的最早完成的任务.

Proof: 用"替换"反证, 记最优解 OPT, 从第一件事开始 inductive

- 假设 OPT 第一件不是最早结束 ⇒ 换成最早结束事件

① 不会与之前冲突(因为没有) ② 不会与之后冲突(因为结束提前了)

- 再设 OPT 前 k 件都是最早结束. 第 k+1 件不是. 那么换.

↓ 取前 k 件后最早开始的事 x

若 x 不是前 k 件后最早结束的事 e, x↔e, e 仍不与前后事冲突.

4. Huffman Code

P_i 升序、取最小两个数 P_0, P_1. 并得 P_0+P_1 插入 order 中

nlogn 实际只需 O(n)

因为每次新插入的位置一定在上次之后. 则总共只需扫一遍

5. Makespan (NP-hard. 但有任意近似 若 ALG ≤ (1+ε)OPT ⇒ $O(poly(\frac{1}{\varepsilon})poly(n))$)

Input: m machines, n jobs with size P_i

Output: minimize finished time. (among all machines)

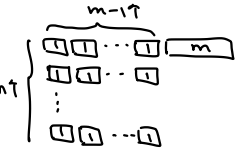
① Greedy 1: 每次随机选一个任务放在最早完成的机器

Approximation Ratio = 2

ALG=2m-1

OPT=m

η = 2-1/m



证明 ALG≤2OPT:

设最晚结束的任务 x, 开始于 t

(可认为 x 也最晚开始, 否则有 x' 比 x 晚开始早结束. 去掉 x' 不影响 ALG, 且 OPT 单减. 可以)

在 t 之前一定所有机器有任务 (因为 x 宁排在这里是因只有它结束)

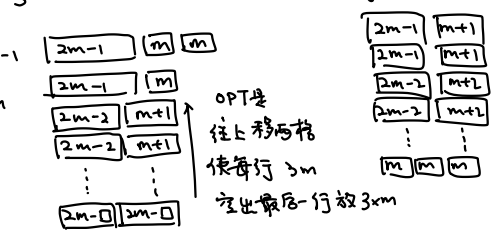
workload ≥ t·m. 则 OPT ≥ t, 而 ALG = t+P_x ≤ OPT+OPT

② Greedy 2: (Longest Processing Time first) 每次选最大放在最先完成

Ratio = 4/3. m~2m-1 的 job 各 2 个; m 的 job 1 个

LPT = 4m-1

OPT = 3m



证明 LPT ≤ 4/3 OPT 同上 LPT = t+P_n ≤ OPT+P_n

(首先易证 $P_n \le \frac{1}{2}$OPT. 因前面任务都比 P_n 大. 且每个机器至少一个

则 OPT 面对至少 m+1 个至少 P_n 大的任务. 最好 OPT 也得 ≥ $2P_n$

⇒ LPT ≤ t+P_n ≤ $\frac{3}{2}$OPT)

特判 若 t=0. 上面推出 t 前任务满不成立, 但此时任务 < m. ALG=OPT

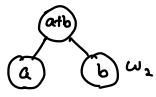
我们希望 $P_n \le \frac{1}{3}$OPT. 但并无法做到 ⇒ 分情况证

① $P_n \le \frac{1}{3}$OPT ⇒ 直接 LPT ≤ t+P_n ≤ $\frac{4}{3}$OPT

② $P_n > \frac{1}{3}$OPT ⇒ OPT < 3P_n. Lemma1 每个机器至多 2 任务. (否则总有一行 > 3P_n)

Lemma2 ($P_1 > \cdots > P_{2m}$) $P_1 \sim P_m$ 必在不同机器上, Lemma3 $P_{m+1} \sim P_{2m}$ 必与前面长短相反 → (否则换) ⇒ Greedy 也这么排. LPT=OPT

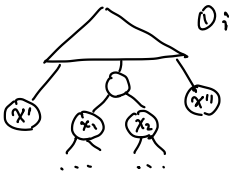
6. Huffman 最优证明（二叉情况下）



将下层的 cost 全挪在上面结点 $w' = aw_1 + bw_2$

则设最终 (a+b) 在 OPT 中位于深度 h. 则 $= \underbrace{w' + (a+b)h}$ 即：将两部分分开

下层算过的 cost 永远不变，变的只是大结点变深时的额外 cost.



① 设下层 x_i 及其子树与 OPT 同结构，则 w' 相同.

② 下一步合并 x_i 中最小 x_1 和 x_2. 反证.

（其余各点要么本层合并，要么与上面的层合并）

若在这层合并 x' 与 x_2，首先合并不影响各 w'

但 x_1 以在其它结点处合并. 至少更浅 $height(x_1) \le height(x')$, $\Delta h \ge 0$.

则交换 x_1 与 x'. $\Delta cost = -\Delta h \cdot x' + \Delta h \cdot x_1 = \Delta h(x_1 - x') \le 0$. cost 减.

Huffman Encode $\Leftrightarrow$ min cost prefix free binary tree.

[补 P2. Super Plan 定理证明 HW2.1]

- If SCC1 reach SCC2. 则 SCC1 最大 finish > SCC2 最大 finish

仍从 P2 "重要思考点" 出发，能 reach 无非两种情况：

在 DFS 中探索了 / 没探索.

① 当 SCC1 到 SCC2 的探索，则 SCC1 的起始点结束时一定完全探索了所有该点能到的点，即包括全体 SCC2 $\Rightarrow$ 该点结束比所有 SCC2 晚

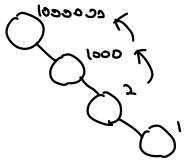
② 当 SCC1 到 SCC2 的探索，但由于 SCC1 能到 SCC2. 则 DFS 时必先 DFS 探索的 SCC2. 于是 SCC1 全体开始都比 SCC2 全体结束晚.

[补 Union-Find Set 复杂度]
- Union $O(1)$ by rank
- Find
  - 查自己：m 次 $O(m)$
  - Charge
    - SGC $O(n \log^* n)$
    - AGC $O(m \log^* n)$

1. AGC (across group)

Kruskal 中 $|E|$ 次 Find（判断 group(u) 时要调用 Find.
中每条边会产生对两个节点的 Find）

Bound：一次 Find 最多 $\log^* n$ 次 AGC. 因为 charge 到之后下一次 charge 要更大的组来 charge，一共就 $\log^* n$ 组



AGC 在描述这种跳增不会太多

2. SGC (Same group)

Lemma 1：准确 rank = k 的点至多 $\frac{n}{2^k}$

Lemma 2：故对于 $[k+1, 2^k]$ 组，至多 $\frac{n}{2^k}$ 个点

$\left(\frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \cdots + \frac{n}{2^{2^k}} \le \frac{n}{2^k}\right)$

Lemma 3：某个点每次 charge 后下次一定被更大 rank charge.

（charge 后甩到根（rank 更大），即 charge 它的有到少单增）

综上：Lemma 3 $\Rightarrow$ 每个点在 group 内至多 SGC $2^k - (k+1)$ 次

Lemma 2 $\Rightarrow$ group 内最多 $\frac{n}{2^k}$ 个点，则组内至多 $\frac{n}{2^k}[2^k - (k+1)]$

$< n$ 次 SGC $\Rightarrow$ 共 $\log^* n$ 组，全部 SGC 最多 $O(n \log^* n)$

[补3 Fibonnaci Heap 均摊势能] 势 $= t + 2m$

m：marked 点数，t：根数，D：最大度数

1. Update $O(m')$ marked 的数量越多，以后 cut 花费越大

若现有 #CC 个 cascating cut. 即说明 #CC 个 marked

则花费 $O(\#CC + 1)$ 来切

对势的影响：$\Delta t = \#CC + 1$ 多了这些根

$\Delta m = -\#CC + 1$ 但清了这些 mark

$\Rightarrow \hat{C} = O(\#CC+1) + \delta(\Delta t + 2\Delta m) = O(\#CC+1+\#CC+1-2\#CC+2) = O(1)$

2. Pop Min $O(t^- + 2D)$ 根越多，以后 pop 花费越大（重新找 min）

pop 后把它的儿子全放藤上，然后合并相同度数 merge.

但最大度数 D $\Rightarrow$ 至多 D 个根（1~D 各 1 个）

$\Rightarrow$ 新 $t^- \le D$. $\Delta t = D - t' \le 0$

$\hat{C} = O(t^- + 2D) + \delta(D - t') = O(D) = O(\log n)$

DynamicProgramming

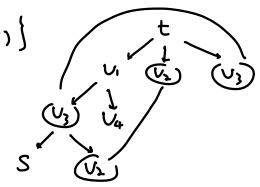
Design: 1. 设计一个 recursive 算法
2. 合并相同问题
3. 找到解决问题的拓扑序 (by hand)
4. Solve & store subproblems in Topo Order

正确性证明关键: 假设的 State 最优一定出现在全局最优中

1. DAG中单源最短路 dist(t) 表示从源到 t 的最短路

$dist(t) = \min_{(v_i,t)\in E} \{ dist(v_i) + w(v_i,t) \}$

若把重复点合并, 就是 G 原图



=> 已知是DAG. 按Topo序求解即可

- Algorithm
  1. Find Topo Order of V: $O(|V|+|E|)$
  2. dist[S]=0
     Solve & record dist[u] in Topo order: (上面转移方程) $O(|V|+|E|)$
- Correctness
  (按 Topo 的顺序 inductive 证). 设 dist[u_i] 均最短 $\forall i<k$.
  dist[u_k] 只能从 $u_1', u_2', \cdots u_m'$ 来 (所有能来的都在他 Topo 序前. 已解)
  求 min 后必最短.

2. 最长递增子列 Longest Increasing Subsequence
$\{a_1, a_2, \cdots, a_n\}$. 子序单增 LIS

State: LIS(k) 表示以 a_k 结尾的 LIS 长度

- 朴素地可以遍历 k 之前的结尾. 取 $a_i < a_k$ 中 LIS(i) 最大的. → $O(n^2)$
- 利用单调队列 — 维护 sm[len]. 达到 len 长度的最小结尾数.

len	0	1	2	3	4	5	6
Smallest	0	a_x	a_y	a_z	-	-	-

下一个 a_i 进来后. 找到 $sm[l] < a_i < sm[l+1]$ → 可二分查到 → 共查 n 个数 $O(n\log n)$

比 sm[l] 大说明以 a_i 结尾至少是 l+1. 比 sm[l+1] 小说明接不到它后面

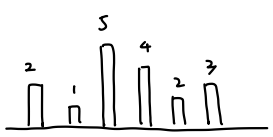
=> LIS[a_i] = l+1. 并且可以替换 sm[l+1] 为 a_i (因 a_i 更小)

=> 输出最大的 l 满足 sm[l] 非 -

(sm[l] 更新 a_i 是从左往右按顺序, 而 a_i 也只能接在前面数. 故正确)

- 单调队列 monotonicty queue { 双端队列: 移最左/右, 插右 + 单调性: 首到尾单减

例:
$\{a_1, \cdots, a_n\}$. 窗口 size = k. 返回所有窗口中的最大值



比如 k=3 (2 (1 5) 4) 2 3

一定不会在任何窗口内最大: 比 5 先离开窗口. 还比 5 小

1. 及时去掉无用数据 (又老又菜): 新来的数据踢掉前面所有比它小的
2. 保证双端队列有序 (pop 离开窗口的): 左端 pop 掉超出窗口的

Time: 由于每个数仅会被 push + pop 一次.
且答案 LargestNumIn[i] = 第 i 次更新时的 front. 取它 O(1)
=> 总和 O(n)

3. 编辑距离

$$ED[i,j] = \min \begin{cases} ED[i-1,j-1] + I(w_i \neq y_j), & \leftarrow \text{认为最后一位对齐} \\ ED[i-1,j] + 1, & \leftarrow \text{S2 补一位对齐} \\ ED[i,j-1] + 1 & \leftarrow \text{S1 补一位对齐} \end{cases}$$

4. 背包 Knapsack (NP-hard)

0/1 背包. 物品只有一个: cost C_i, value V_i, capacity W.
Maximize $\sum V_{select}$ (不可分割物品)

若 f[i] 定义为前 i-1 个物品在 W 下的最优解 => 则在能买的情况下总是想全买 ✗

子问题定义太弱 => 增加 dp 一个维度
导致难以递推 直到空间所有状态可互相转移

增加一维: f[i,w] 定义为 前 i-1 个物品在给定 w 下的最优解

此时状态易转移: 可以分类每个物品买/不买

转移: $f[i,w] = \min \{ f[i-1, w-C_i] + V_i, f[i-1,w] \}$
(买 i / 不买 i)

(若 $C_i > w$ 当前预算买不起, f[i,w] 直接 = f[i-1,w])

- Time Complexity: $O(nW)$. → 注意 W 并非输入规模.
输入规模是 bit 数. 设 Input n 件物品. 则表达它们 C_i, V_i 至少要 2n bits, 那不妨再输入 n bits 制造一个 2^n 大小的 W
=> 在 Input Size = n 的情况下. W 可达 2^n => $O(n2^n)$ 非 poly
- 空间优化: 不用必须 O(nW). 可用一维数组递推
f[w]: max value can get with w budget.
$f[w] = \max_{i=1\sim n} \{ f[w], f[w-C_i] + V_i \}$ 即把所有物品一遍比完
- Surplus Knapsack; Complete Knapsack.

5. Minimize Manufacturing Cost
$\{a_1, \cdots a_n\}$. 需要按顺序分组做完所有, $Cost(l,r) = C + (\sum_l^r a_i)^2$
(若 C=0. 对分组无惩罚. 则每次做单任务最优; 若一次方, 则只分一组最优)

状态 f[i]: 完成前 i 件事的最小 cost.

$f[i] = \min_{j<i} f[j] + C + (\sum_j^i a_-)^2$ 遍历 a_i 与前面 n 件事为一组

f[0]=0. 朴素遍历 $O(n^2)$

优化思路: 找真中单调的成分 potential XX → 首先研究什么是 potential 好

考虑什么情况分组截到 y 比截到 x 好 $(a_x \cdots (a_y \cdots a_i)$

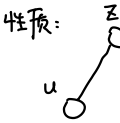
记 S(i) 为 $a_1+a_2+\cdots+a_i$

截到 y: $f[y] + C + (S(i)-S(y))^2$. 截到 x 同.

当 y better 时: $S(i) > \frac{(f[y]+S^2(y)) - (f[x]+S^2(x))}{2(S(y)-S(x))}$

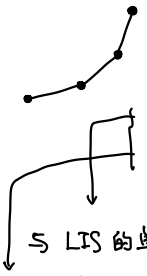
=> 每个任务看作点 $(2S(x), f[x]+S^2(x))$. x,y 点斜率 < S(i) ⇒ 右侧点更好

性质:



当左侧斜率 > 右侧, z 一定没用, 无论 S(i) 是多少
① 若 $k_{uz} > S(i)$. 则 u 比 z 好
② 若 $k_{uz} < S(i)$. 则 $k_{vz} < k_{uz} < S(i)$. v 比 z 好.

由性质可以去除所有 ↗ 的点. 剩下斜率递增的 Convex Hull



计算新来的 f[i] 实际就是从头遍历比较斜率与 S(i) 出现 $k_{x,x-1} < S(i) \le k_{x+1,x}$ 时停

去掉之前 x 个点. 因 S(i) 单↑, 必定也比以后 S(i') 小

f[i] 最小值要从 x+1 处递推过来 (以从 x+1 ~ i 构成一组)

⇒ LIS 的单调队列完全相同的操作. 每个点仅入一次、出一次.

算完 f[i] 后引入新点, 重新从最后一组点开始踢非 Convex Hull 的点

- Time Complexity: 更新 ConvexHull 与踢人是联系在一起的.

每踢一个点, 才会计算 $(2S[i], f[i]+S^2[i])$ 与下一个旧点的斜率.

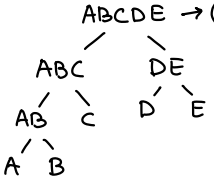
选最小值也一样. 比较过斜率便踢掉. 不会再比了. $O(n)$

6. Product of Sets

$L_1, L_2 \cdots, L_n$ 按顺序做笛卡尔积. 求最小乘积数.

比如 $(L_1 \times L_2) \times L_3 = m_1m_2 + m_1m_2m_3$; $L_1 \times (L_2 \times L_3) = m_2m_3 + m_1m_2m_3$

⇒ 本质: 是计算树的切分



则 f({A,B,C,D,E}) 可由比该集合少的 f(S) 推来. $S \subseteq \{A,B,C,D,E\}$

只需要遍历 n-1 长度"隔板"位置即可

⇒ 总的来说. 是一个二维 dp. f[i,j] 表示子集起始.

- Time Complexity: 朴素需要 $O(n^2)$ 个状态, 转移 $O(n)$, 则总 $O(n^3)$

F.Frances Yao 证明: dp 状态满足平行四边形法则的, 可优化至 $O(n^2)$

Quadrangle Inequality: $\forall i \le i' \le j \le j'$: $w(i,j)+w(i',j') \le w(i',j)+w(i,j')$

Monotonicity: $\forall i \le i' \le j \le j'$: $w(i',j) \le w(i,j')$

7. 最短路的 DP 视角.

- Bellman-Ford (单源 S→所有) dist[k,u] 从 S→u 的 k 条边路径最短长度

```
for k = 1 to |V|
  for each (u,v)∈E
      dist[k,v] = min{ dist[k-1,v], dist[k-1,u] + d(u,v) }
```

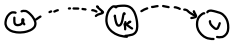
- Floyd-Warshall (多源 all pair)

dist[k,u,v]: u 到 v 的最短距离, 只经过 $U_1, U_2, \cdots, U_k$ 中的点

提供了很好的转移性质:

k-1 → k 的转移只比较两个状态 { 不经 U_k / 经过 U_k

不经: dist[k-1,u,v]

经过:  dist[k-1,u,Uk] + dist[k-1,Uk,v]

算法: 1. dist[0,u,v] = d(u,v) for all (u,v)∈E. (otherwise ∞)

2. for k=1 to |V|: $O(|V|^3)$

for u = 1 to |V|:

for v=1 to |V|:

dist[k,u,v] = min{ dist[k-1,u,v], dist[k-1,u,Uk]+dist[k-1,Uk,v] }

(可用 dist[u,v] = min{ dist[u,v], dist[u,k]+dist[k,v] } 将空间降至 $O(|V|^2)$)

8. 树上 Max Independent Set

$$f(x) = \max\left\{ \underbrace{\sum_c f(c)}_{\text{不选 } x},\ \underbrace{1 + \sum_{cc} f(cc)}_{\text{选 } x} \right\}$$

状态 f(u): 以 u 为根的子树的最大独立集 size

- Time: 有 n 个状态, 即每个状态至多被检查 2 次: 一次作儿子、一次作孙子

⇒ 总时间 $O(n^2)$

- Greedy: 从叶子考虑. 若 OPT 选了倒数第二层. 则换成它的所有孩子(叶子)不破坏 OPT. 且叶子 ≥ 倒数第二层

若 OPT 没选倒数第二层, 则选上叶子. ⇒ 倒二一定不选. 叶子一定全选

实现: 一次 DFS: 到叶子则选上(标绿), 返回把父亲标红.

如果回溯后发现所有孩子标红 → 标绿. ⇒ 一次 DFS $O(|V|)$ 完成

- 为什么可以求和——关键在于各子树之间独立.

则各子树间选好 independent set 后, 连起来必还是独立的

Tree Composition —— 描述图的"像树性质" ⇐ 设计一个树兼拓 ⇓ 的算法

{ Cycle: 2 / Clique: n-1 / Tree: 1 / ⋮

分解成各个 bag { 1. 包含同一个点的 bag 之间是连通的 (即所有含 u 的 bags 连通成一个分量)

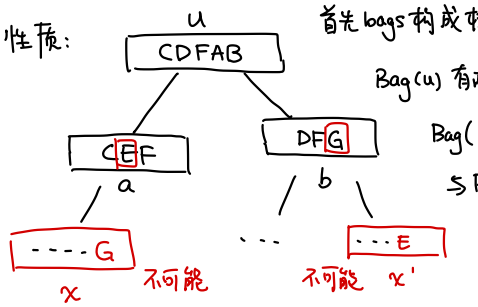
2. 存在 (u,v) 的边 则 u,v 在同一个 bag 中

Treewidth = Maxsize(bag) - 1.

(若 G 为树. 则 Treewidth = 2-1 = 1.

即每条边 没一个 bag 包含两侧的点)

△性质:



首先 bags 构成树. 无 cross edge

Bag(u) 有两子树 T(a) 与 T(b)

Bag(T(a)) - Bag(u)

⊇ Bag(T(b)) - Bag(u)

独立

a 中有的、不在 u 中的点. 必然与 b 中有的、不在 u 中点 独立.

因为若 a 连接的后面某层的 bag x 中含 G. 由性质 1.

x 与 b 均含 G. 它们应连通. 但 u 将子树分隔开

(u 是 a,b 之间唯一通路)

解: 状态 f(S,u), 表示 bag u 中选且仅选 S 集合中的点

Topological Order: 自叶向上.

叶子: $f(S,u) = \begin{cases} |S|, & \text{if } S \text{ independent} \\ 0, & S \text{ not available.} \end{cases}$

树上 dp 转移: $f(S,u) =$

$$|S| + \sum_c \Big(\max_{\substack{S' \in B(c) \\ S' \cap B(u) = S \cap B(c)}} f(S',c) - |S \cap B(c)| \Big)$$

$S' \cap B(u) = S \cap B(c)$: 枚举在 B(c) 中未被 S 决定是否选择的其余所有情况.

$|S \cap B(c)|$: 共有的点已经有 S 中记数过了

小部分枚举. 大部分之间 dp

Network Flow

1. Ford-Fulkerson Method — residual net 剩余网络想法

Flow 严格定义： $f: E \to R^+$，定义域 $e \in E$.（即将边映射到实数）

- Capacity Constraint: $\forall e \in E.\ f(e) \le C(e)$.
- Flow Conservation: $\forall u \in V \setminus \{s,t\}$ $\sum_{(u,u)\in E} f((v,u)) = \sum_{(u,w)\in E} f((u,w))$ 除 s,t 外所有入=出

- FF 核心：剩余网络 $G^f(V^f, E^f)$：$V^f = V$ 一样，任意两点邻接就有边
  - 若 $(u,v)\in E$，$C^f(u,v) = C(u,v) - f(u,v)$ 正向为向，E^f 中记录剩多少 capacity
  - 若 $(v,u)\in E$，$C^f(u,v) = f(v,u)$ 反向为向，E^f 中反向记录 flow 了多少

① Initialize f: $f(e)=0$. $G^f \leftarrow G$.

② While $\exists$ s→t path on G^f:
  find minimum capacity b edge on path. ← 每次更新路上最少处，即一次更新至阻塞
  for each $e=(u,v)\in$ path:
    if $(u,v)\in E$: $f(e) \leftarrow f(e)+b$ 使用了正边，增加流.
    if $(v,u)\in E$: $f(e) \leftarrow f(e)-b$ 使用了反边，减少流
  Update G^f ← 更新边的 E^f 中边权值 正边 C-f，反边 f

③ return f.

Correctness: Max-Flow-Min-Cut 定理

将图分为两部分 L（含S），R（含t）

- Lemma0: 流 $V(f) = f_{out}(L) - f_{in}(L)$

$$f_{out}(L) - f_{in}(L) = \sum_{u\in L} f_{out}(u) - f_{in}(u) \quad \leftarrow \text{拆成各个点的和，成立因为内部抵消.}$$
$$= f_{out}(s) + \sum_{u\in L\setminus\{s\}} f_{out}(u) - f_{in}(u) \quad \leftarrow \text{把 s 情况单算}$$
$$= V(f) + 0 \quad \text{（由入=出约束为 0）}$$

- Lemma1: For any flow f and cut $\{L,R\}$. $V(f) \le C(L,R)$

$$V(f) = f_{out}(L) - f_{in}(L) \le f_{out}(L) = \sum_{\substack{u\in L\\ v\in R}} f(u,v) \le \sum C(u,v) = C(L,R)$$

从左向右的流 ≤ 这些边的权重和

而所有 $u\in L, v\in R, (u,v)\in E$ 的总和必然构成了割

- Lemma2: $\exists$ cut $\{L,R\}$ 与 Ford-Fulkerson 给出的 flow $V(f)$ 相等

将 L 定义为 在 G^f 中可由 S 到达的总集，剩余为 R.

① $f_{out}(L) = C(L,R)$，否则说明从 L→R 的边未满，则在 G^f 中必然可由 L 到达 R.

② $f_{in}(L) = 0$ 即从 R 中无向 L 的流，否则 G^f 中仍有 L→R 反流.

综合两点 $V(f) = f_{out}(L) - f_{in}(L) = C(L,R)$

⇒ 所有 $V(f) \le$ 所有 $C(L,R)$. 而 FF 给出的 flow 与某个 $C(L,R)$ 相等

⇒ FF 给出的 Flow 是最大 $V(f)$.

Algorithm for finding s-t Min-Cut：以 S,t 作为源、目标建图跑 M-F，capacity=边权 ⇒ 得出最大 flow 后，在 G^f 中可由 S 到达的边为 L，R=V\L.

正确性由 Max-Flow-Min-Cut 定理保证.

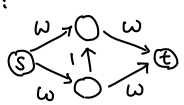
这种不断维护（类似迭代）的算法，要证两部分
① Optimal, if halt
② Can halt → 并且这会和复杂度一起算.

Time Complexity: —— 若随机找 S→t 或用 DFS 找 $O(|E|\cdot f_{max})$

① 边权 integer：由于每次找到 S→t 路并更新后，总 flow 必增加（因为 $f_{out}(S)$ 肯定增加，$V(f) = f_{out}(S)$.）故至多 $\frac{f_{max}}{1}$ 轮停止

- 最差可达到 f_{max} 轮循环（即最差每轮只增了 1）

Bad Case:



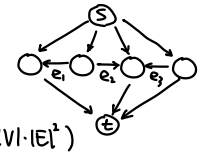
$w \gg 1$，f_{max} 显然为 2w

但若每次 S→t path 选取随机，可能总先找中间边，导致每次只增 1.

② 边权有理数：scale 后整数（本质是总有加加减减最小单位）

③ 边权实数，若 DFS 找路，实数可能不会停

$e_1 = e_3 = 1$. $e_2 = \frac{\sqrt{5}-1}{2}$（满 $e_2^2 = 1 - e_2$）



2. Edmonds-Karp —— BFS 找 S-t path $O(|V|\cdot|E|^2)$

"While $\exists$ S→t path in G^f" 这一步用 BFS 找.

BFS 关键：在 G^f 中，找的 S→t path 长度 non-decreasing.

因为每次加反边，而反边不会减少距离（反边是从后向前指）

对每个点都有 dist(u) 不减，—— 但不 Strictly increase

⇒ 虽然 dist(t) 只有 1, 2, …, |V|, +∞ 共 |V|+1 个取值，但在每个值上有未知次数停顿.

⇒ 考虑什么东西的增长是严格单调的：边成为 critical 每次至少一条

每次找 path 上阻塞处，称这条边 (u,v) 为 critical.

$dist(v) \ge dist(u) + 1$. 若 (u,v) 以后还要成为 critical，必然要先使用 (v,u)，则在那一轮中 $dist'(u) \ge dist'(v)+1 \ge dist(v)+1 \ge dist(u)+2$

⇒ dist(u) increases 2 between two critical

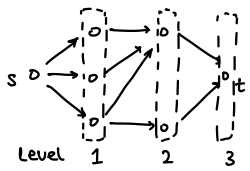
平均而言一次 critical 增 1，至多被 critical $\lfloor \frac{|V|}{2} \rfloor \times 2 + 1 = O(|V|)$ 次

⇒ 共 $O(|E|)$ 条边，每次循环有一条边 critical，至多 $O(|V||E|)$ 循环

⇒ 每次循环 BFS $O(|E|)$，则总时间 $O(|V||E|^2)$

3. Dinic — 引入 level graph $O(|V|^2|E|)$

Level Graph:



用 BFS 构建 LG，花费 $O(|E|)$

第 i 层表示 dist(u) = i

- 只保留 i 层 → i+1 层的边（即去除回边和层内边）

Algorithm:

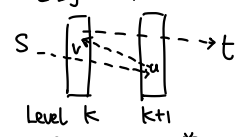
① 在 LG 上找 block flow：所有 S→t path 必须至少一条边 critical

也即将所有能到 S→t 的路都堵死了

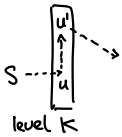
在 LG 上 DFS 找到 t 路两种可能：
- 找到 S→t path：找 critical 边并设置 flow，并删 critical 边
- 没找到，中途遇到 block 处（即 capacity 满）：则删掉 block 点的所有入边.

则每次 LG 上的 DFS 花费 $O(|V|\cdot|E|)$：因为每条路走到头至多 $O(|V|)$（到 t 或到 dead-end），而一次至少删一条 critical 边

② 注意到一次循环后必从 LG 中 k+1 层生出一条新边 {指回 K 层 / 或在 k+1 层内部



指回 K 层：dist(t) 增加 2.

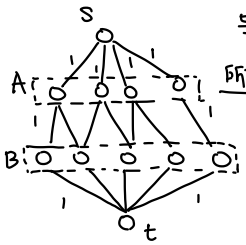


在 K+1 层内部：dist(t) 增加 1

dist(t) 严格单增

至多 $O(|V|)$ 轮循环，Dinic Time: $O(|V|^2|E|)$

4. 网络流应用1：二分图匹配（Matching 定义为 edges that 不重复涉及点，二分图上 matching 就是配对问题）



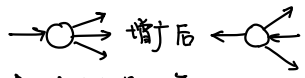
与普通网络流有本质区别：

所有边权是1. 则每次跑无论 block 还是找到 path 都必然删掉整条路.

⇒ 每轮循环中至多 $O(|E|)$ 便停

每条边只会往过一次.（反边也一样边权1）

- Claim 1: 每次至多 $|E|$ 轮找全 block flow
- Claim 2: $V\backslash\{s,t\}$ 的入度与出度至少有一个是 1

（Inductively. 二分图初始满足，  增广后 入也与出边交换，各自数量不变）

△ edge/vertex disjoint 边/点不相交路

① 找最多的 edge disjoint 数量 from S to t：（用网络流找 edge disjoint）

构造 S→t 的网络流，并设边权都是1.

在最终停止 G^f 中看从 S 出了 n 条边

② 由 Claim 2：每个点至少入度出度有一个为 1.

故不会出现多个 edge disjoint path 经过同一个点

则 点不相交路数量 = 边不相交路数量（二分图例中. vertex disjoint = edge disjoint）

- Claim 3: 至多 $2\sqrt{|V|}$ 轮循环

—— 证：经过 $\sqrt{|V|}$ 轮后，G^f 的最大流至多 $\sqrt{|V|}$

首先 最大流 = 边不相交路数，再由 claim 3，也 = 点不相交路数

于是我们有 $\sqrt{V}$ 轮后 { s-t 最短路 $\geq \sqrt{|V|}$ 长度
⇒ $|V|$ 个点至多 $\sqrt{|V|}$ 个点不相交路

⇒ G^f 此后最大流至多 $\sqrt{|V|}$

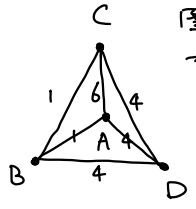
⇒ $\sqrt{|V|}$ 轮可停止.

⇒ 每轮循环 $O(|E|)$，共 $O(\sqrt{|V|})$ 轮

⇒ Hopcraft-Karp-karzanov（二分图上跑 Dinic）$O(|E|\sqrt{|V|})$

5. 网络流应用2：锦标赛胜场

	胜场
A	4o
B	38
C	37
D	2P



图表示任两队剩余场次，求 D 是否有机会赢下比赛.

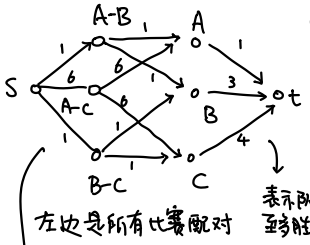
=>

首先让D全胜

D为41胜

⇒ A 至多 win 1，B 至多 win 3，C 至多 win 4

构建网络流



左边是所有比赛配对

右边是队伍.

流量表示有几场比赛

表示队伍至多胜场

若 max-flow 能让左侧流满，则存在分配

① 有胜场 ⇒ flow：只需要按比赛结果分流，赢者分到一个流，总能分完左侧场次.

② 有 flow ⇒ 实际胜场：

定理：若边 capacity 均整数，必有整数流，且必可由整数达到

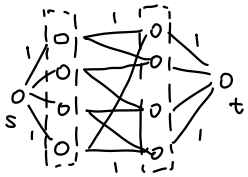
由 FF 过程一开始更新 Critical 边必为整数，后面一直加减整数，流必为整数

⇒ 整数达到的流可分配为实际胜场

6. 定理1：regular 二分图必有 perfect matching

↓ (∀ vertex. same degree)　↓ (all vertices matched)

{ 构造一个全边权整数的网络流 ⟶ 证明最终可有整数分配
{ 算出该图可达满射的整数流 ⟶ 有完美匹配的条件



正常建图，所有边权1.

若给每个A类点向每个B类点发出 $\frac{1}{d}$ 的量是满足 Capacity 的

⇒ flow 可达 $|A|$，且这是一个割大小，就是最大流

⇒ 由于边权整数，该 max-flow 必有整数 flow 达到 ⇒ 有完美匹配

7. 定理2：Hall's Marriage：

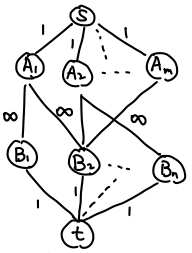
If $|S| \leq |N(S)|$ for all $S \subseteq A$ ⇔ 存在 matching size $|A|$（左侧全配）

① "⇐" 显然，否则 $\exists S$. $|S| > |N(S)|$，那这 S 个人必不可能匹配上.

② "⇒" 构建网络，该网络满足右侧性质.

欲证有 A→B 全映射，要找到 $|A|$ 的网络流

⇒ 证明该网络最小割 = $|A|$



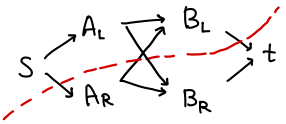
中间边权没为1也行 ∞ 便于证明

Proof：将割分成三类

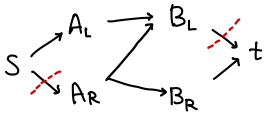
1. 直接切 Ⓢ，割 = $|A|$

2. 直接切 Ⓣ，割 = $|N(A)| \geq |A|$

3. 若割如下所示，部分割 Ⓢ，部分割 Ⓣ，分 $\{S, A_L, B_L\}\{A_R, B_R, t\}$ 两组



严格说应该这样切是割：切 $|A_R|$，切 $|A_L| \to |B_R|$，切 $|B_L|$



但若割中有 $|A_L| \to |B_R|$ 直接 ∞ 3，故正常割的图中不存在 $A_L \to B_R$ 边

故切掉的边只剩两侧：

割 = $|A_R| + |B_L| = |A_R| + |N(A_L)| \geq |A_R| + |A_L| = |A|$

⇒ 所有割都大于等于 $|A|$

而最多匹配 $|A|$ 条，说明 max-flow 就是 $|A|$

Linear Program

1. 形式

$$\max\ c_1x_1+c_2x_2+\cdots+c_nx_n$$
$$\text{s.t.}\ a_{11}x_1+a_{12}x_2+\cdots+a_{1n}x_n \le b_1$$
$$\vdots$$
$$a_{m1}x_1+a_{m2}x_2+\cdots+a_{mn}x_n \le b_m$$
$$x_1,\cdots,x_n \ge 0$$

性质：
① always $\exists$ optimum $(x_1,\cdots,x_n)$ 为这个约束多胞体 (polytope) 的顶点

2. 单纯形法 —— 在顶点游走找最优
最差指数 (遍历所有顶点) 时间，但在高斯扰动下 (smooth analysis) 仍near bound 在多项式 (run fast in practise)
(椭球法、内点法，均理论bound多项式)

3. 标准形：
① max ② 约束 $\le$ 若原 $=$ 则加松弛 原 $\ge$ 则 $x \to -x$
③ x_i 无 ≥ 0 约束，换成 $x_+ - x_-$，$x_+, x_- \ge 0$

4. 网络流 → LP (Ford-Fulkerson 就是 simplex 算法，只不过选方向很聪明)
变量是 f_{uv} for all $(u,v)\in E$

$$\max \sum_{(s,u)\in E} f_{su} \qquad \text{s.t.}\ 0 \le f_{uv} \le C_{uv},\ \forall (u,v)\in E$$
$$\sum_{(v,u)\in E} f_{vu} = \sum_{(u,w)\in E} f_{uw},\ \forall u \in V\setminus\{s,t\}$$

5. Duality 理解

$$\max\ x_1+6x_2+13x_3$$
$$\text{s.t.}\ x_1 \le 200$$
$$x_2 \le 300$$
$$x_1+x_2+x_3 \le 400$$
$$x_2+3x_3 \le 600$$
$$x_1,x_2,x_3 \ge 0$$

各乘 $y_1\sim y_4$

希望凑出一个最小的值
$x_1+6x_2+13x_3$
$\le (y_1+y_3)x_1+(y_2+y_3+y_4)x_2+(y_3+3y_4)x_3$
$\le 200y_1+300y_2+400y_3+600y_4$

由于 $x_1,x_2,x_3\ge 0$，目标变为最小化这个值
则要求 $y_1+y_3\ge 1$，$y_2+y_3+y_4\ge 6$，$y_3+3y_4\ge 13$

$$\max\ c^Tx \quad \text{s.t.}\ Ax\le b,\ x\ge 0 \quad \Longleftrightarrow \quad \min\ b^Ty \quad \text{s.t.}\ y^TA\ge c^T,\ y\ge 0$$

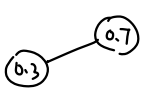
强对偶声明 LP 问题 prim OPT = dual OPT.

6 LP-relaxation 将 Integer LP $\xrightarrow{\text{放松要求}}$ LP (ILP即多一行 $x\in\mathbb{Z}^n$)
relax 为 $x_i\in[0,1]$.
最后转换回满足 integer 要求的解直接取 round.

例：Vertex Cover relax后 ILP → LP 为：
(本身是 $x_v\in\{0,1\}$ 代表选/不选)

$$\min \sum_{v\in V} x_v$$
$$\text{s.t.}\ x_u+x_v\ge 1 \quad (u,v)\in E$$
$$0\le x_v\le 1 \quad v\in V$$

算法结果对每条边会有分数的预测



⇒ 一个近似处理：把分数中 $\ge\frac12$ 的点都选上

2 近似：证：首先 relax 后一定可以更优，在 min 下即更小 $OPT(IP)\ge OPT(LP)$

$$\text{而 } OPT(LP)=\sum x_v=\sum_{x_v<\frac12}+\sum_{x_v>\frac12} \ge 0+\sum_{x_v>\frac12}\cdot\frac12$$
$$=ALG\cdot\frac12$$

数量即我们选的点数

$$\Rightarrow ALG \le 2OPT(LP) \le 2OPT(ILP)$$

- VertexCover ↔ Matching (König-Egervary 定理)

Dual (relax 后)：

$$\max \sum_{e\in E} x_e$$
$$\text{s.t.}\ \sum_{(u,v)\in E} e_{uv}\le 1,\ \forall v\in V$$
$$e_{uv}\ge 0$$

(相当于每个点被选择量不能超过 1 次，match了不能再match)

7. MaxFlow ↔ MinCut 用对偶证明该定理

$$\min\ \sum y_{uv}C_{uv}+0,\ (u,v)\in E$$
$$\text{s.t.}\ y_{su}+z_u\ge 1,\ (s,u)\in E$$
$$y_{ut}-z_u\ge 0,\ (u,t)\in E$$
$$y_{uv}-z_u+z_v\ge 0,\ (u,v)\in E,\ u\ne s,v\ne t$$
$$y_{uv}\ge 0,\ (u,v)\in E$$

(z_u 无约束因为原本约束③是=0约束)

(这里 y_{uv} 的意义是 (u,v) 是否为最小割一部分
z_u 的意义是最小割中 u 在 S 还是 t 侧)

证明过程：
1. 问题1与问题2对偶 (ILP，但写出的是relax后LP)
2. 证明其中一个问题有整数解 (By 系数矩阵Total unimodular)
3. 由 Strong duality 知两个问题最优解相同 (若无上一步，又能说明 LP 的 opt 相同，但ILP不一定，而2说明ILP与LP解相同)
4. 构造解，证明符合 primal 约束 (比如是整数 match，但有的边，有的±2)

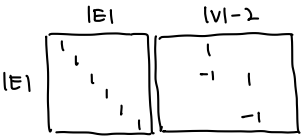
定理：If $A\in R^{m\times n}$ is total unimodular ($\det$(任意子式) $=0,\pm1$)
且 b 是主整数向量 ⇒ polytope $P=\{x: Ax\le b\}$ 都是 integer vertices
⇒ 进一步据单纯形法，找到的 OPT 也必然整数

Proof Sketch：考虑取其中若干约束构成的方程边界 $A'x=b'$
由 Cramer's rule：$x_i=\dfrac{\det(A_i'|b')}{\det(A_i')}$ → 为 ±1 而 b' 也整数 ⇒ x_i 整数

下面证明 max flow 的约束矩阵满足 total unimodular

共有 $|E|$ 条约束，变量共 $|E|+|V|-2$ 个

约束矩阵：



左侧也的变量都必为 1
因为约束中每条也对应且仅对应 f_{uv}.
右侧，每行要么 1 个数要么 2 个数 (1 或 -1，1 和 -1)

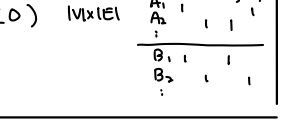
Lemma 1：单位阵 total unimodular
Lemma 2：A 是 total unimodular ⇔ A^T，$[A|I]$，$\begin{bmatrix}A\\I\end{bmatrix}$ 也是
证：

选出部分 I 后，用定义展开
每次取 I 的一个 1，消去一行一列
一直迭完 I 中的单独 1..
最后剩下 $(-1)^k\cdot\det$(A子式)

Lemma 3：该 $|E|\times(|V|-2)$ 的矩阵是 total unimodular.

Inductively：1×1 的子阵都满足；假设任意 $k\times k$ 子阵都满足
现在选取任一 $(k+1)\times(k+1)$ 的子阵：
性质：① 其每行要么只有一个数 ±1，② 要么只有两个数 1 和 -1
若整个子阵存在一行①：则取这行，
$\det(A_{k+1})=\pm1\times\det(A_k)\in\{-1,0,1\}$ ✓
若整个子阵不存在①，即每行都有一个 1 和 -1，
则把所有列加到第一列，$\det$ 不变，而第一列变成了全 0
⇒ 此情况下 $\det(A_{k+1})=0$，⇒ 仍是 total unimodular

(消去的方法依题而变，根据图本身/矩阵本身的性质，例如二分图中考虑所有边均有两个非零元素的情形 (都1)，由于每边必连一个A一个B类，则把 A 类的列 ×(-1) 加所有 B 类的列 ⇒ 全0) $|V|\times|E|$



• 于是由 lemma3 和定理，这个 LP 可由整数还到
下一步要从给定的整数解中恢复出 max flow 的边.

作如下分割： $L=\{s,\ z_v\geq 1\text{的}\}$ $R=\{t,\ z_u\leq 0\text{的}\}$

由 primal-dual 的关系 $cut(L,R)\geq flow(f)$.

现欲证 $C(L,R)\leq \sum C_{uv}y_{uv}$ 便能得出等于.

显然：因为 $C(L,R)$ 的每一项都是某条边 C_{uv}. y_{uv} 表示是否是割-部分则左侧加 C_{uv}. 右侧也加 C_{uv}.

$$C(L,R)=\sum_{\text{某}\uparrow\text{割}} C_{uv} \leq \sum_{\text{某}\uparrow\text{割}} y^*_{uv}C_{uv} \leq OPT(dual)\leq C(L,R)$$

$$\Rightarrow MinCut = C(L,R) = OPT(dual) = OPT(Prim) = MaxFlow.$$

8. Zero-Sum Game

A 操作集 $\{a_1,\cdots,a_m\}$. B 操作集 $\{b_1,\cdots,b_n\}$

零和：对每个操作对 (a_i,b_j) AB 的收益和为 0：$u_A(a_i,b_j)+u_B(a_i,b_j)=0$

Payoff Matrix（只需一个来描述，因为零和） G_{ij} 为 A 在 ↘ 的 gain
或 B 在 ↘ 的 loss.

例：Rock-Scissors-Paper

$$G=\begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}$$

Von Neumann's Minimax 定理：
Zero Sum 中，谁先决定策略最后最优解相同

记 A 的一系列 strategy $x=\{x_1,\cdots,x_m\}$ B 的 $y=\{y_1,\cdots,y_n\}$

$u_A(x,y)=x^TGy$ $u_B(x,y)=-x^TGy=-\sum G_{ij}x_iy_j$

若 A 先出策略，则每种策略 x 必会被 y 针对

A 要找到被针对下最优 x： $\max_x \min_y \sum G_{ij}x_iy_j$

同理 B 先出时，要找所有 y 被某个 x 针对下最优 y： $\min_y \max_x \sum G_{ij}x_iy_j$

Lemma 1：最优解必 pure strategy

因为展开 $\max_x \min_y\ y_1\sum G_{i1}x_i + y_2\sum G_{i2}x_i + \cdots + y_n\sum G_{in}x_i$

⇒ 实际就是优化到最小的 $\sum G_{ik}x_i$ 上，令 $y_k=1$. 其余 $y=0$

$$\Rightarrow \max_x \min_{j\in[1,n]} \sum G_{ij}x_i$$

写成 LP 形式：

$$\max_x\ z$$
$$s.t.\ \sum G_{ij}x_i - z\geq 0 \quad \forall j=1,\cdots,n$$
$$x_1+\cdots+x_m=1 \quad (归一化\ distribution)$$
$$x_1,\cdots,x_m\geq 0$$

Dual：

$$\min\ w$$
$$s.t.\ \sum G_{ij}y_j\leq w \quad \forall i=1,\cdots,m$$
$$y_1+\cdots+y_n=1$$
$$y_1,\cdots,y_n\geq 0$$

⇒ 正是对 y 优化的 $\min_y \max_x \sum G_{ij}x_iy_j$

再由 Strong duality ⇒

Minimax Theorem： $\max_x \min_y \sum G_{ij}x_iy_j = \min_y \max_x \sum G_{ij}x_iy_j$

NP

1. P/NP 问题定义

P: polynomial TM (solver). A(x) 直接给结果

NP: polynomial TM (verifier). A(x, y) 多给个提示 (验证简单, 搜答案难) 然后判断 x,y 是否合法

2. Karp Reduction

- 3SAT 作为 SAT 特例. 显然更简单 3SAT $\leq_K$ SAT
- 但若有 SAT 问题输入 X. 经 Poly Time 转成 X'. $SAT(X)=\begin{cases}1, & 3SAT(X')=1\\ 0, & 3SAT(X')=0\end{cases}$

则 SAT $\leq_K$ 3SAT (说明若有3SAT的verifier, 也可以验证SAT问题)

=> SAT $\leq_K$ 3SAT. 进一步说明两问题"同样难"

构造: $(x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4 \vee x_5 \vee \neg x_6)$ → 引入中间变量, 代表左/右变量取值情况

$(x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee \neg x_3 \vee y_2) \wedge (\neg y_2 \vee \neg x_4 \vee y_3) \wedge (\neg y_3 \vee x_5 \vee \neg x_6)$

3. NP-hard / complete

- A decision problem f is NP-hard: $\forall g \in NP.\ g \leq_K f$
- NP-complete: $f \in NP$ and f is NP-hard.

4. Cook-Levin 定理: All NP $\leq_K$ SAT

所有 NP 问题 (存在多项式时间Verifier的问题) 都可归约到 SAT.

思路:

Verifier是一个图灵机 → 想办法把状态转换翻译成 SAT.

	x_1	x_2	x_3	----	x_n	y_1	y_2	----	y_m	S	P
TM 运行	0	1	0	..	1	1	1	...	0	0	0
多项式步骤	0	1	1	...	1	1	0	...	0	1	1
	⋮										
	1	1	0		1	0	1	...	1	1	0

Verifier 的功能是: 若对于某输入 X 是 valid 的. 则 $\exists Y$

使得 Verifier(X, Y) = 1.

在TM上来说, 就是第一行输入 X,Y 的二进制 ①, 最后一行是OK的二进制 ②

且这张表的中间部分每行演化遵循 TM State 转移规则. ③

①②③ 可分别用 SAT 布尔变量表达 —— 每个格子设为变量. 记录 0/1

5. Reduction Map (→ 均指代 $\leq_K$)

SAT → 3SAT ① → Independent Set ② → Vertex Cover ⑥ → Dominating Set

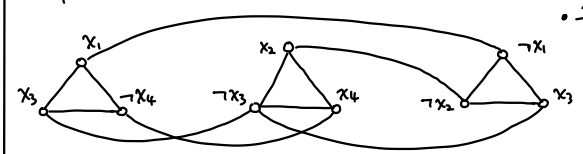
Independent Set ③ → Clique

Vertex Cover ④ → Subset Sum ⑤ → Partition

3SAT ⑦ → 有向 Hamilton Path ⑧ → 无向 Hamilton Path ⑨ → Hamilton Cycle

① 3SAT $\leq_K$ IS

$\phi = (x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$



- 每个子句构造一个三角形
- 正反变量连接起来

证明 3SAT(X) 与 IS(X') 是 Yes⇔Yes, No⇔No:

- 若 3SAT(ϕ, {x_i}) 输入一组取值 Yes, 即每个子句至少有一个点. => 构造X'时. 在每个三角中任选一个 True 的点 (若有单三角形内多True, 必然总总数 > k)

⇒ IS(G, {x_i'}, k). 若这些选点符合 ϕ. 则必然满足 Size为k 的 IS. ⇒ Yes

- 若 3SAT(ϕ, {x_i}) 是 No. 则必然有一个 clause 全 False ⇒ 构造X'时. 这个三角里也要任取一个. 那么必然和前面该变量的逆布尔 ⇒ 不是IS ⇒ No

② IS $\leq_K$ VC 若 S 是 G 的独立点集, 则 V-S 必是 G 的点覆盖.

任两点之间无边 ⟹ 选/不选调换 ⟹ 任两没选的点之间无边. 则必然选了的点 cover 所有边.

IS(G, S, k) ⇒ VC(G, V-S, |V|-k)

③ IS $\leq_K$ Clique (团: 内部完全图)

若 S 是 G 的独立点集, 则 S 是 $\bar{G}$ 中的团 (补图是原来两点间有边/无边调换)

任两点之间无边 ⟹ 原来无边现都加上 ⟹ 任两点之间有边. 即完全图.

IS(G, S, k) ⇒ Clique($\bar{G}$, S, k)

④ VC $\leq_K$ SS: $A=\{a_1, \dots a_n\}$. k 判断是否 $\exists S \subseteq A$, $\sum S = k$

借用一个中间问题 Vertex Cover $\leq_K$ Vector Subset Sum

本质是一个特殊的 VSS, 可再转化到 $\leq_K$ Subset Sum

(Vector Subset Sum: 将 A 中的元素数字改成向量, k也改成向量. 判断是否有 $\sum \vec{v_i} = \vec{k}$, 即要求向量每位都对上)

思路: 图上与选向量的共同点在于"选/不选"

=> 希望将 Vertex Cover 里点的选/不选与向量选/不选对应

VC: $x_v \in \{0,1\}$

s.t. $\begin{cases} x_v + x_u \geq 1. & (u,v) \in E \\ \sum x_i = k \end{cases}$

相应设置 |V| 个向量, 每个 |E|+1 长度

第1到0/1表示选不选这个点

如果选的话:

对应第 i 个点 cover 了哪些边.

为了把 $x_v + x_u \geq 1$ 这个不等式子转化成"求和等于". 引入松弛变量 y_{uv}

=> $x_v + x_u + y_{uv} = 2$.

① 第一列求和为 k. 表示 VC 大小为 k, Vi 里要选 k 个

② 后面 |E| 列求和为 2. 若只有一个 Vi 为 1. 则下面要选择相应边. 否则不用选相应边

V_1(1, 0, 0, 1, ----)

V_2(1, 0, 1, 1, ---)

V_3(1, 0, 0, 0, ---)

⋮

全1 若选该点便计数

e_1(0 1

e_2(0 1 (单位阵)

⋮ ⋱ 1

K(k, 2, 2, ----- 2, 2)

=> VC $\leq_K$ VSS

而这个 VSS 有性质 { 均非负. 且除第一列外每列总和 ≤ 5 (1+1+2) }

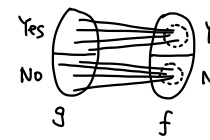
=> 可以将每行看作十进制数. 直接求和的目标 SS 也能做.

=> VSS $\leq_K$ SS. 于是借由 VSS 得证 VC $\leq_K$ SS.

Proof Sketch of NP-Complete

1. 证 NP
2. 先选一个 NP-complete g. 证 $g \leq_K f$
3. 证明等价性: 对你信息的 Input 转换方式.

{ 若 g(X) = Yes, 则 f(X') = Yes;
若 g(X) = No, 则 f(X') = No (或若 f(X) = Yes, 则 g(X) = Yes) }



Lec21. P12 证明

所以只证 f(x') = Yes ⇒ g(X) = Yes 是不够的. 还有 g(x) = Yes 对应到 f 中.

(当然. 这个方向简单)

⑤ SS $\le_k$ Partition: $S=\{s_1,\cdots s_n\}$. 判断是否有分隔使 $\sum_{a\in A} a = \sum_{b\in B} b$

$\le_k$ Partition$^+$: 只能判断正数分隔了

- 首先对于 Partition 的转化：将原输入数组增加一个 $2k-\sum_{i=1}^{n} a_i$

$SS(\{a_1,\cdots,a_n\},k) \Rightarrow Partition(\{a_1,\cdots,a_n,\ 2k-Sum\})$

则若前者为 Yes，有 $\sum a' = k$，则后者必有分割 $\sum a' = k$，剩余也是 k

则后者有一个分割，不妨设将 $a_1,\cdots,a_n$ 分为 x 与 $Sum-x$.

则无论 $x+(2k-sum) = sum-x$ 还是 $x = sum-x+(2k-sum)$，均有一部分为 k.

- 对于 Partition$^+$，可由 Partition reduce 过来.

$Partition(\{s_1,\cdots s_n\}) = Partition^+(\{|s_1|,\cdots,|s_n|\})$

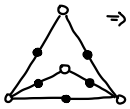
若 $\{s_1,\cdots,s_n\}$ 有分割，必可写成 $P_1-n_1 = P_2-n_2$ (positive & negative)

则在绝对值中，必有另一分割 $P_1+n_2 = P_2+n_1$.

反之绝对值有分割原数组也必有 (本质是正数与负数部分名自差值相同)

⑥ VC $\le_k$ DS: For any vertex $u\in V\setminus S$. $\exists u\in S$. $(u,v)\in E$.

类比 Vertex Cover 覆盖边，希望转化为覆盖点 (注意 VC 是覆盖边)



⇒ 在每条边上额外加点，若覆盖了这些点，也就代表覆盖了边

但这不够：因为 VC 要求"覆盖所有边"只等价于要求"覆盖所有边中点"，也许没覆盖完原点也行，但 DS 就不承认.

⇒ 将原点们补成完全图，保证了原点均被覆盖

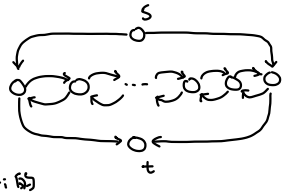
- 有 VC 的 Yes ⇒ DS 的 Yes：VC 首先 cover 所有边 ⇒ cover 所有新加点；由完全图，也必 cover 所有原点
- DS 的 Yes ⇒ VC 的 Yes：把所有选了的新加点任意换成两端的某个原点

新加点换成原点，cover 的边数一定不降.

⑦ SAT $\le_k$ 有向图 Hamilton Path 存在性.

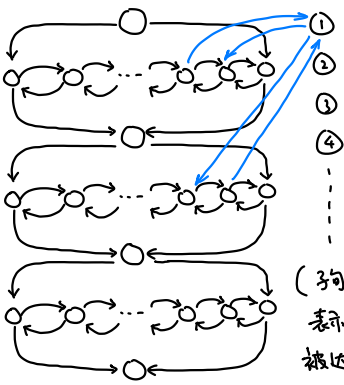
对于：$(x_1\vee x_2\vee\neg x_3)\wedge(\cdots$

给每个变量构造一个结构



性质：1. 从 S 起始走完的唯二方法是先走左和先走右

2. 中间圈的数量是包含 x_i 的子句数量. 下面利用性质 1. 给每个子句中的 x_i 或 $\neg x_i$ 添加描述.



规定该结构是向左走的则 x_i = True

比如对①的子句 $(x_1\vee\neg x_2)$

为了达到①，要么 x_1 结构从左向右走 (x_1 = True)

要么 x_2 结构从右向左走 (x_2 = False).

(注意到每出去一次会用掉一个点 故每个子句要造一个圈)

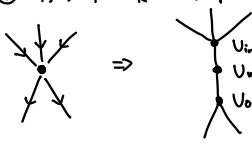
(子句数个点，表示每个子句都需要被达到/实现)

- SAT 的 Yes ⇒ 有 HP：依照赋值决定每个圈左/右走，由保证了每个 clause True，则必有其中一个值为 True，则从这个变量的圈可达该子句.

⇒ 所有子句可达 ⇒ Hamilton Path

- 有 HP ⇒ SAT 的 Yes：依照 path 每个圈左/右走来赋值变量即可.

⑧ 有向 HP $\le_k$ 无向 HP



把一个点拆成三个. ⇒ 避免了从原图中的入边进入边出的情况

(1) 有向 HP ⇒ 无向 HP. 直接走对应边即可

(2) 无向 HP ⇒ 有向 HP.

性质：Uin 必接上一个的 Uout，Uout 必接 Uin

由性质，若无向 HP 有一组点顺序为 Uin-Umid-Uout 或 Uout-Umid-Uin.

则所有组顺序都是这个 ⇒ 对应有向图中正路或反路. ⇒ 也必存在

⑨ 无向 HP $\le_k$ Hamilton Cycle.

大致：图上 → Independent Set；数值相关 → Subset Sum；否则 → SAT

Approxiate Algorithm (假设是 minimize 问题)

1. ratio 定值 $ALG(Input) \le \alpha\cdot OPT(Input)$ $\forall$ Input.

2. ratio 关于 n 增长不断变差，比如 $\alpha(n) = O(\log n)$

n=100 时 $ALG \le \log 100\cdot OPT$

n=10000 时 $ALG \le \log 10000\cdot OPT$

3. PTAS & FPTAS (Polynomial Time Approximation Scheme)

多项式时间逼近任意常数近似比

PTAS: $\alpha = 1+\varepsilon$. 则这个 scheme 可达 $O(n^{\frac{1}{\varepsilon}})$

Fully PTAS: $O((\frac{1}{\varepsilon})^c n^a)$

4. FPT (Fixed-Parameter Tractable)

精确求 OPT，但不一定多项式.

$O(f(k)\cdot n^c)$ k 是一个描述输入情况有多差的参数.

即对于 NP-hard 问题设计一个含参数的算法，比如 Independent Set 问题"像树程度"