

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
- 有些章节写得很啰嗦，有些则很简略，因为笔记产生的初衷并不完全为了
Cheat Paper，更大的原因是很喜欢老师讲课的思路，故花费了一些时间
把老师所讲的全部分析样例都总结了下来，算是对为数不多的高质量课程的
尊重和留念

- 笔记没有添加水印，如果你愿意，你可以直接打印为 Cheat Paper
(但以后学期可能会更新一些分析样例)
- 但经验表明 Cheat Paper 几乎不会被使用——本门课程的重点在于分析
方法而非样例本身，如果不真正理解分析方法，期末考试还是有相当难度的
- 因此我的初衷是这个笔记仅作为“我的理解”抛砖引玉，希望给你复习过
程中在一些问题上提供没准另一个思考角度

Intro

1. Halt Problem : $\text{Halt}(\text{TM}, y) = \begin{cases} 1, & \text{if Turing Machine } (y) \text{ halt} \\ 0, & \text{not halt.} \end{cases}$

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

因为可以 def $B(x)$:

if $\text{Halt}(x, x) = 0$: return 1
else: for(i);

$B(B)$ Halt 的分支
要求 $B(B)$ 不 Halt

即 $B(B) \rightarrow \text{判断 Halt}(B, B)$ $\xrightarrow{\text{=0的分支要求}} B(B) \text{ not halt}$

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

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

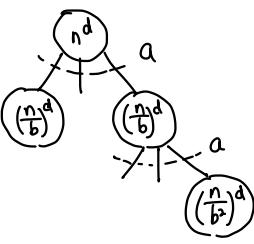
$\Theta(g(n))$. $T(n) = \Theta(g(n)) = \Omega(g(n))$

Divide

1. Master Theorem

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

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



2. Karatsuba

$$\begin{aligned} x \cdot y &= (a \cdot 10^{\frac{n}{2}} + b) \cdot (c \cdot 10^{\frac{n}{2}} + d) \\ &= ac \cdot 10^n + (ad + bc) \cdot 10^{\frac{n}{2}} + bd \\ &= ac \cdot 10^n + [(a+b)(c+d) - ac - bd] \cdot 10^{\frac{n}{2}} + bd \end{aligned}$$

要计算 ac, ad, bc, bd
④ 问：分治后还是 n^2
但 $ad + bc$ 整体等于
 $(a+b)(c+d) - ac - bd$
三次乘法

$$T(n) = 3T\left(\frac{n}{2}\right) + C \cdot n \Rightarrow T(n) = O(n^{\log_3 3})$$

若位数不是 2^k , 则左侧补零

3. ① Divide — into small subproblems

② Recurse — solve subproblems recursively

③ Combine — combine outputs of subproblems

④ Basic Solver

4. Merge Sort

$$T(n) = 2T\left(\frac{n}{2}\right) + C \cdot n$$

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 $3^{k(n)}$ 种情况, 但总排序情况 $P_n = n!$

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

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 $\{x < v\} \cup \{x = v\} \cup \{x > v\}$ $O(n)$

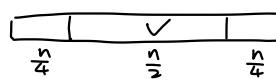
if $k \leq |L|$: return $\text{Select}(L, k)$

elif $|L| < k < |R|$: return v

elif $|L| + |M| < k$: return $\text{Select}(R, k)$

问题在于: 若每次 pivot
都靠近边缘,
则只减少了长度 1
 $O(n) + O(n-1) + \dots = O(n^2)$

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



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

$$\therefore T(n) \rightarrow T\left(\frac{3n}{4}\right) + C \cdot n$$

若每次都能选在 V 区, 则 $T(n) = T\left(\frac{3n}{4}\right) + C \cdot n \Rightarrow O(n)$
但实际上第 k 次选在 V 区是二项分布. $P = \frac{1}{2}$

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

$$\therefore E[T(n)] = E[T(\text{选到V区}) + T(\frac{3n}{4})] = Cn + E[T(\frac{3n}{4})]$$

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

④ 平衡选 pivot 时间和好处 — 若 random 选很快, 但可能差
这样选 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 f'nv: $T\left(\frac{n}{5}\right)$ 即 $\text{Select}(M, \frac{|M|}{2})$

比 $\frac{3n}{10}$ 大 \leftarrow 故 pivot 一定在 $[\frac{3n}{10}, \frac{6n}{10}]$ 区间内.

比 $\frac{3n}{10}$ 小 分成的三段最差最长 $\frac{7n}{10}$

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

内部该证明: 猜 $T(n) \leq B \cdot n$

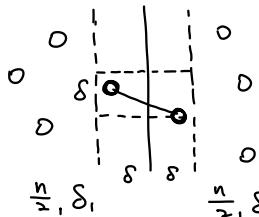
$$T(1) = 1 \leq B \cdot 1$$

$$T(n) = T(0.7n) + T(0.3n) + Cn \leq 0.9Bn + Cn = Bn$$

$$\Rightarrow T(n) \leq Bn \text{ 归纳得证.}$$

错误: 归纳不能用 $O(n)$ 记号, 必须写出 $C \cdot n$ 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 $O(\frac{n^2}{\delta})$

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

\Rightarrow 至多比较 7 次. (7 不重要)

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

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

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. 除先排好 (每次要 S' 的顺序就遍历总顺序)
用一次 $O(n \log n)$ 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 $T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n)$ 再加领先 $O(n \log n)$

7. 多项式乘法 — 应用 FFT

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

多项式 $A(x) = a_0 + a_1 x + \dots + a_n x^n$, 可以只记系数 $\vec{a} = [a_0, i, a_n]$

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 $\hat{C}_j = \hat{a}_j \cdot \hat{b}_j$

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

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. $p(\omega^0), p(\omega^1) \dots p(\omega^{n-1})$

$$\text{注意 } p(z) = p_0 + p_1 z^1 + p_2 z^2 + p_3 z^3 + p_4(z^2) + p_5 z^5 + p_6(z^2)^3 + \dots \\ = P_{\text{even}}(z^2) + z \cdot P_{\text{odd}}(z^2), \text{ 其中 } P_{\text{even}} \text{ 是取一半系数的多项式}$$

$\Rightarrow n$ 个点需要算 P_{even} on $\omega^0, \omega^2, \omega^4, \dots \omega^{2n-2}$
 P_{odd} on $\omega^1, \omega^3, \omega^5, \dots \omega^{2n-1}$

这仍是 $2 \times \frac{n}{2}$ 个插值. 关键①: $\omega^0 \sim \omega^{n-1}$ 平方之后是 西周童合 即变成 (ω^2) 为单位根绕 2 圈. 只要算 $2 \times \frac{n}{4}$ 个插值. 再花 $O(n)$ 合并

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \Rightarrow T(n) = O(n \log n)$$

关键②: $\omega^0, \omega^2, \omega^4$ 再平方后是以 ω^4 为单位根.
 \Rightarrow divide 可以一直做到底.

Graph

一、基础

$$1. G(V, E). 0 \leq |E| \leq |V|(|V|-1) = O(|V|^2)$$

$$2. \text{临接矩阵} \text{ space } O(|V|^2)$$

$$\text{临接表} \text{ space } O(|V|+|E|)$$

3. def explore(v):

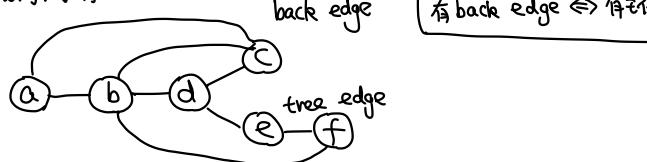
```
marked[v] ← true
for each (u, v) ∈ E
    if marked[u] == false: explore(u)
```

→ 一次独立 explore 可以找全一个连通分量. 但图不一定连通.

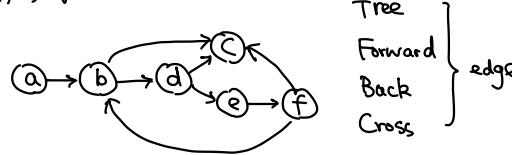
4. DFS 不停更新 explore:

```
def DFS(G):
    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



有向图的 DFS tree



5. topological Order

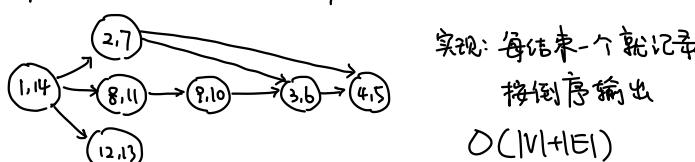
有向无环 DAG \Leftrightarrow 存在 topological order

" \Leftarrow " 易反证
 若有环必无序

" \Rightarrow " 通过构造 DAG 构造拓扑序来证

① DAG 必有 tail ② delete tail \Rightarrow new DAG ③ delete new tail...
 然而每次找 tail 太重复了. $\Rightarrow O(|V|^2)$

Improve by DFS: \Rightarrow start/finish \Leftrightarrow DAG 没有 back edge



```
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 $\text{finish}[v] > \text{finish}[u]$, no $(u \rightarrow v)$

proof: 枚举所有三种可能的边类型 (no back)

① 若 $u \rightarrow v$ 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 $u \rightarrow v$ 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.

\Rightarrow 所以 v 必定完全完成后才进行 u . $\text{finish}[v] < \text{start}[u]$

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

• finish time 的降序就是拓扑序

6. Strong / weak Connect Component

↓ 改成无向后的 CC

$\forall u, v \in \text{SCC}$: u, v 能互相到达, 且是 maximal CC

每次都要从 "tail SCC" 中的某个点开始 DFS. 否则跨 SCC.

最晚结束的点 ($\text{finish}[u]$ 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.
 因此最晚结束应在宝体的 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 $\exists v \in \text{SCC2}, x \in \text{SCC1}$.
 有路径 $v \rightarrow x \rightarrow \dots \rightarrow u$

矛盾: 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.

\Rightarrow 有路径 $u \rightarrow \dots \rightarrow v$. 存在环 \Downarrow .

Basic 实现: 每次 DFS 新的反图, 逐次去生成新图: $O(n(n+m)) = O(nm)$

Super 实现: 先 DFS 一次存所有点的 finish time,
 接降序依次 DFS, 总共一次完整 DFS \Rightarrow 2:2 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 $T\left(\frac{3n}{4}\right) + Cn$.

至 $\frac{1}{2}$ 的情况可认为什么都没做 $\Rightarrow T(n)$.

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

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

```
def BFS(G,S):
```

i < 0

$V_0 \leftarrow \{S\}$ 第0层的点

while V_i is not empty:

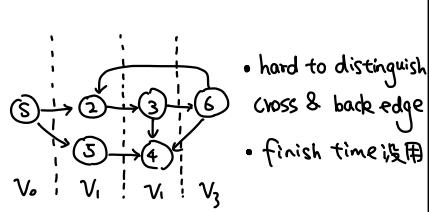
for each $u \in V_i$:

for each $(u,v) \in E$:

if $\text{marked}[v] = \text{false}$:
 $\text{marked}[v] = \text{true}$
add v to V_{i+1}

i++

$\boxed{\text{pre}[v] = u}$

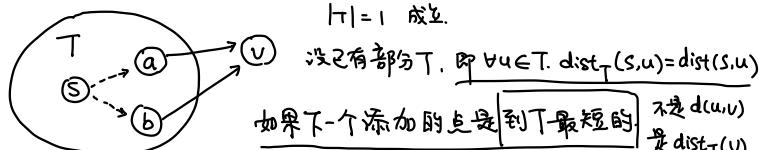


- hard to distinguish cross & back edge
- finish time 没用

2. Short Path tree — 从根S到HU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



即 $\text{dist}_T(S, u) = \min_{u \in T} \{ \text{dist}_T(u) + d(u, u) \}$ 是最短的 u

此时若想经过其它非T中点到达U, 第一步 $d(u, x)$ 就 $> d(u, U)$ 若真的 $|T| \rightarrow x \rightarrow U$ 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U都最优, 但 SPT

$\{S, x_1\} \rightarrow \{S, x_2, x_3\} \rightarrow \{S, x_2, x_3, x_4\}$

不是添加“新边最短的”

而是 $\text{dist}_T(U)$ 最短的 U , 即只用T最短的

def Dijkstra(G,S): 单源 无权

1. Initialize

$T = \{S\}$.

$\text{tdist}[S] = 0$, $\text{tdist}[v] = \infty$ for other vertices

$\text{tdist}[v] = d(S, v)$ for $(S, v) \in E$ ← 这也是 update

2. Explore

$|V|$ rounds

find $v \notin T$ with min $\text{tdist}[v]$

$T \leftarrow T \cup \{v\}$ ↑ 为了维护 $\text{tdist}[v]$ 时刻是从

3. Update $\text{tdist}[v]$ ↑ 当前 $\{T\}$ 能抵达U的最短路

$\text{tdist}[u] = \min \{ \text{tdist}[u], \text{tdist}[v] + d(u, v) \}$

$|E|$ rounds for $(u, v) \in E$

只用新加的点 v 与原来比较来更新

因为每条边只会

产生 2 次 update

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

↓
find min + update, 以后不找了 某个结点变小了. 重新维护

d-array

$O(d \log d)$

$O(\log d)$

$O(\log n)$

$O(n)$

Binomial

$O(\log n)$

$O(1)$

$O(\log n)$

$O(\log n)$

Fibonacci

$O(\log n)$

$O(1)$

$O(1)$

$O(1)$

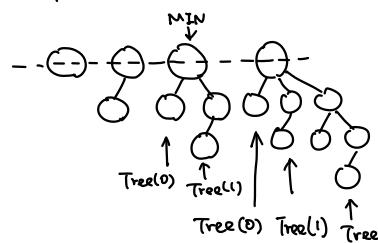
① 若用数组. $T(n) = O(|V||V| + |E|) = O(n^2)$

② 若 2 叉堆 $T(n) = O((|V| + |E|) \log |V|)$ 稳定 $O(n^2 \log n)$

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 $0 \sim k-1$ 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 pop_{\min} 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. $D \leq \log n$.

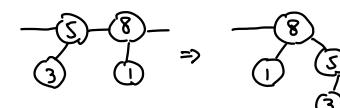
④ Cascading Cut Rule: 每棵树至多可失去一个 child.

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^-

⑤ Merge Rule: 每次新生成很多根后, 将相同度数合并.

由 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 $\Rightarrow D \leq \log n$



• 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 \Rightarrow 均摊

• Potential Function

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

$$\Sigma \hat{C} = \Sigma C + \delta \cdot \Phi, (\text{能均摊要求至少即 } \Sigma \hat{C} \geq \Sigma C)$$

\Rightarrow 均摊 $O(\frac{\Sigma \hat{C}}{N})$ 操作次数 即必须向差收敛.

• Update: $O(m) \rightarrow$ 需合并次数

$$\Delta \Phi = t + 2m$$

#CC Cascading cuts, #CC marks, #CC roots

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

$$\Delta t = \#CC + 1$$

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

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

因为 m 会多造成

cascading cut 以及潜在
多一次 merge

• 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 $\text{dist}_T(x) < \text{dist}_T(u)$, 但可能 $\text{dist}_T(x) + c(x,u) < \text{dist}_T(x) < \text{dist}_T(u)$

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):

$$\text{dist}[S] = 0, \text{dist}[x] = \infty \text{ for other } x \in V$$

while dist is updated:

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

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

每次更新所有边.

- 正确性两步 ① 停止时是 SPT ② 一定会停止

Lemma1: k 轮更新后, $\text{dist}(v)$ 是至多 k 长路的最短路长

Inductive: 若所有 u_{k-1} 都最短. $\xrightarrow{(5)} \cdots \xrightarrow{u_{k-1}} \xrightarrow{v}$

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

Lemma2: $|V|$ 长的路一定比 $|V|-1$ 长的路优

因为 $|V|$ 有环, 则不是这个环一定更优. \Rightarrow 必在 $|V|$ 轮停止

\Rightarrow 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

① 使 $\sum_{j \leq i} s_j \geq d_i$. 其中前 i 个任务满足 $\forall j < i, d_j < d_i$:

\Rightarrow 无法在前 d_i 时间内完成 $i \sim n$ 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 \rightarrow 边无权且包含所有点 $\rightarrow |V|-1$ 条 \rightarrow 树.

Prim: 和 Dijkstra 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 $\Rightarrow |E|$ round Update, $|V|$ Pop min

\Rightarrow Use Fibonacci heap $\mathcal{O}(|E| + |V| \log |V|)$

Kruskal: 将边排序, 挑升序依次选, 若成环则不选

① Sort Edges $\mathcal{O}(E \log E)$ (忽略这个瓶颈, 没 ordered)

② for each $(u,v) \in E$ in 升序: $\leftarrow 2|E|$ round check.

if $\text{group}(u) \neq \text{group}(v)$:

choose (u,v)

union ($\text{group}(u), \text{group}(v)$) \leftarrow total $|V|$ round.

Use Union-Find Set.

Union: $\mathcal{O}(1)$; Find (查 group) 的摊来说 $\left\langle \begin{array}{l} \text{Same Group C: } n \log^* n \\ \text{Cross Group C: } m \log^* n \end{array} \right\rangle$

\therefore 总时间 $\mathcal{O}((m+n) \log^* n)$

均摊原理: rank 小的点很多, 大的点少 \Rightarrow 将小点分散分组.

$[1, 2] [2, 2^2] [2^2, 2^3] \dots$ 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 , 则 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: 完成数量最多)

贪心: 从高到低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 \Rightarrow 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 件后最早开始的事义

若 x 不是前 k 件后最早结束的事义, $x \leftrightarrow e$, 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 . 并将 $p_0 + p_1$ 插入 order 中

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 $\Rightarrow O(\text{poly}(\frac{1}{\epsilon}) \text{poly}(n))$)

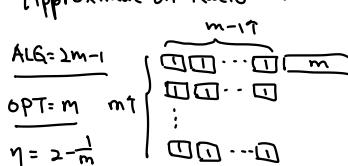
Input: m machines, n jobs with size p_i

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

证明 $\text{ALG} \leq 2 \text{OPT}$:



设最晚结束的任务 X, 开始于 t
(可以为 X 也最晚开始, 否则有 X' 比 X 跟早结束, 去掉 X'
不影响 ALG, 且 OPT 减少, 可以)

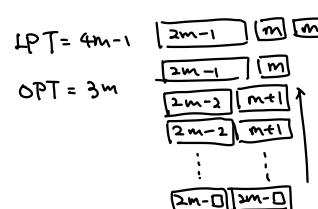
在 t 之前一定所有机器有任务

(因为 X 安排在这里是因为它结束)

$\text{workload} \geq t \cdot m$, 则 $\text{OPT} \geq t$, 而 $\text{ALG} = t + p_x \leq \text{OPT} + \text{OPT}$

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

Ratio = $\frac{4}{3}$. $m \sim 2m-1$ 的 job 各 2 个; m 的 job 1 个



证明 $LPT \leq \frac{4}{3} OPT$ 由上 $LPT = t + p_n \leq OPT + p_n$

(首先易证 $p_n \leq \frac{1}{2} OPT$. 因前面任务都比 p_n 大, 且每个机器至多一个
则 OPT 面对至少 $m+1$ 至少 p_n 大的任务, 最好 OPT 也得 $\geq 2p_n$
 $\Rightarrow LPT \leq t + p_n \leq \frac{3}{2} OPT$)

特别: 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 $p_n \leq \frac{1}{3} OPT$. 但并无法做到 \Rightarrow 另情况论证

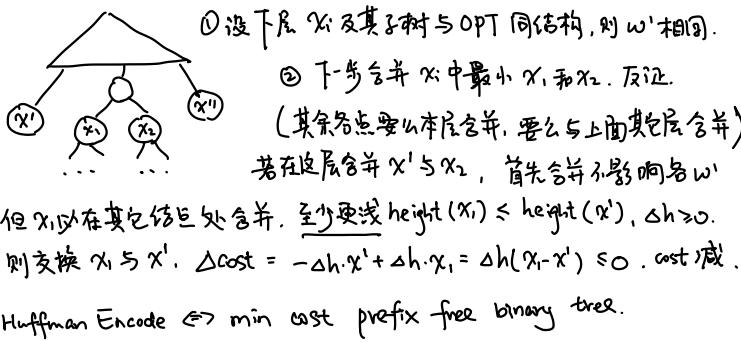
① $p_n \leq \frac{1}{3} OPT \Rightarrow$ 直接 $LPT \leq t + p_n \leq \frac{4}{3} OPT$

② $p_n > \frac{1}{3} OPT \Rightarrow OPT < 3p_n$. Lemma1 每个机器至多 2 任务. (否则有)

Lemma2 ($p_1 \dots p_m$). $p_1 \sim p_m$ 必在不同机器上. Lemma3 $p_{m+1} \sim p_{2m}$ 必与前面
长短相接 \rightarrow (否则接) \Rightarrow Greedy 也这么排. $LPT = OPT$

6. Huffman 最优证明(二叉情况下)

将下层的 cost 全摊在上面结点 $w' = aw_1 + bw_2$. w_1 (a) w_2 (b) w'
则设最终 (a+b) 在 OPT 中位高度 h. 则 $= \frac{w'}{h} + (a+b)h$ 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.



[补 P2. Super Plan 定理证明 HW2.1]

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

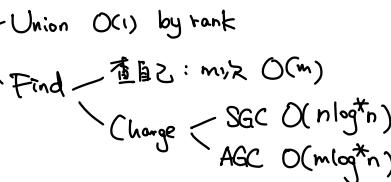
仍从 P2 “重要思考点”出发. 能 reach 无非两种情况:

在 DFS 中探索了 / 没探索.

① 从 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 从 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.

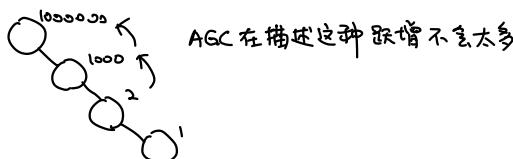
[补 Union-Find Set 复杂度]



1. AGC (across group)

Kruskal 中 $|E| \geq \text{Find}$ (判断 group(u) 时要调用 Find.
而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 要更大的值来 charge, 一共有 $\log^* n$ 组



2. SGC (Same group)

Lemma 1: 该结点 rank = k 的点至少 $\frac{n}{2^k}$

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

$$\left(\frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \dots + \frac{n}{2^k} \leq \frac{n}{2^k} \right)$$

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 它的序列必单增)

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 $\frac{n}{2^k} [2^k - (k+1)]$

$< n \geq 2 \text{ SGC} \Rightarrow$ 共 $\log^* n$ 组. 全部 SGC 最多 $O(n \log^* n)$

[补 3 Fibonacci Heap 均摊势能] $\Theta = t + 2m$

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

若有 $\#CC$ 个 cascading cut. 即说明 $\#CC \uparrow$ marked

则花费 $O(\#CC+1)$ 来切

对 t 的影响: $\Delta t = \#CC + 1$ 多了这些根

$$\Delta m = -\#CC + 1 \quad \text{但消了这些 mark}$$

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

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. 然后会和相同度数 merge.

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

$$\Rightarrow 新 t^- \leq D. \quad 4t = D - t^- \leq 0$$

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

Dynamic Programming

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)

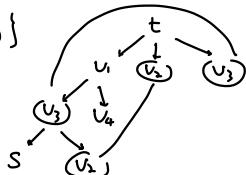
4. Solve & store subproblems in Topo Order

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 $\text{dist}(t)$ 表示从 s 到 t 的最短路

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

若把重复合并, 就是 G 原图



\Rightarrow 已知是 DAG, 按 Topo 序求解即可

• Algorithm

1. Find Topo Order of V : $O(|V|+|E|)$

2. $\text{dist}[s] = 0$

Solve & record $\text{dist}[u]$ in Topo order: (上面转移方程) $O(|V|+|E|)$

• Correctness

(按 Topo 的顺序 inductive 证). 设 $\text{dist}[u_i]$ 均最短 $\forall i < k$.

$\text{dist}[u_k]$ 只能从 u'_1, u'_2, \dots, u'_m 来 (所有能来的都在他 Topo 序前, 已解)
找 min 必最短.

2. 最长递增子列 Longest Increasing Subsequence

$\{a_1, a_2, \dots, a_n\}$. 严格单增 LIS

State: $\text{LIS}(k)$ 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 $\text{LIS}(i)$ 最大的. $\rightarrow O(n^2)$

• 利用单调队列 — 维护 $\text{sm}[l:m]$. 达到 len 长度的最小结尾数.

$len \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$

Smallest $0 \quad a_x \quad a_y \quad a_z \quad - \quad - \quad -$ 可二分查到

下个 a_i 进来后, 找到 $\text{sm}[l] < a_i < \text{sm}[l+1]$ \rightarrow 查 n 个数 $O(n \log n)$

比 $\text{sm}[l]$ 大说明以 a_i 结尾至少是 $l+1$, 比 $\text{sm}[l+1]$ 小说明接不到它后面

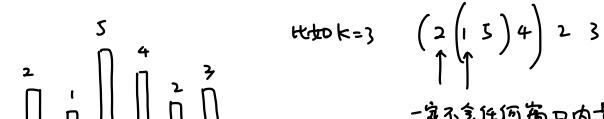
$\Rightarrow \text{LIS}[a_i] = l+1$. 并且可以替换 $\text{sm}[l+1]$ 为 a_i (因 a_i 更小)

\Rightarrow 输出也最大的 l 满足 $\text{sm}[l] \neq -$

($\text{sm}[l]$ 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 monotonic queue } 双端队列: 移最近/右, 插右
单调性: 首到底单减

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



一定不含任何窗口内最大值:
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 $\text{LargestNumIn}[i] =$ 第 i 次更新时的 front. 即它 $O(1)$

\Rightarrow 总和 $O(n)$

3. 编辑距离

$$\text{ED}[i, j] = \min \{ \begin{aligned} &\text{ED}[i-1, j-1] + I(y_i \neq y_j), \\ &\text{ED}[i-1, j] + 1, \leftarrow S2 补一位对齐 \\ &\text{ED}[i, j-1] + 1 \end{aligned} \} \leftarrow S1 补一位对齐$$

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 $\sum V_{\text{select}}$ (不可分割物品)

若 $f[i]$ 定义为前 $i-1$ 个物品 \Rightarrow 则在能买的的情况下
在 W 下的最优解
总是想全买

子问题是定义太弱 \Rightarrow 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

$$\text{转移: } f[i, w] = \min \{ \begin{aligned} &f[i-1, w-C_i] + V_i, \\ &f[i-1, w] \end{aligned} \} \downarrow \begin{matrix} \text{买} \\ i \\ \downarrow \\ \text{不买} \end{matrix}$$

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

• Time Complexity: $O(nW)$. \rightarrow 注意 W 并非输入规模.

输入规模是 bit 数. 设 Input n 件物品. 则表示它们 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

\Rightarrow 在 Input Size = n 的情况下. W 可达 $2^n \Rightarrow O(n2^n)$ 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.

$$f[w] = \max_{i=1 \dots n} \{ f[w], f[w-C_i] + V_i \} \text{ 即把所有物品一趟比完}$$

• Surplus Knapsack; Complete Knapsack.

5. Minimize Manufacturing Cost

$\{a_1, \dots, a_n\}$. 需要按顺序分组做完所有. $\text{Cost}(l, r) = C + (\sum_i^r a_i)^2$

(若 $C=0$. 对分组无惩罚. 则每次做单任务最优; 若一次, 则只分一组最优)

状态 $f[i]$: 完成前 i 件事的最小 cost.

$$f[i] = \min_{j < i} f[j] + C + (\sum_j^i a_j)^2 \text{ 遍历 } a_i \text{ 与前面几件事为一组}$$

$f[0] = 0$. 朴素遍历 $O(n^2)$

优化思路: 找其中单调的成分 potential XX \rightarrow potential 好

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

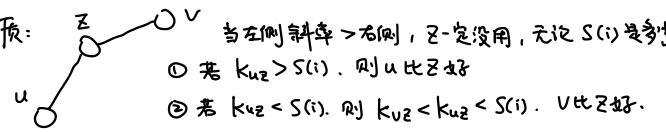
记 $S(i)$ 为 $a_1 + a_2 + \dots + a_i$

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

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

\Rightarrow 每个任务看作点 $(2S(x), f[x] + S^2(x))$. x 离斜率 $< S(i) \Rightarrow$ 右侧点更好

性质:



① 若 $k_{U2} > S(i)$. 则 U 比 V 好

② 若 $k_{U2} < S(i)$. 则 $k_{U2} < k_{U1} < S(i)$. V 比 U 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 $k_{x_i, x_1} < s(i) \leq k_{x_{i+1}, x}$ 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 $x+1 \sim i$ 构成一组）

与 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 Convex Hull 与踢人是联系在一起的。

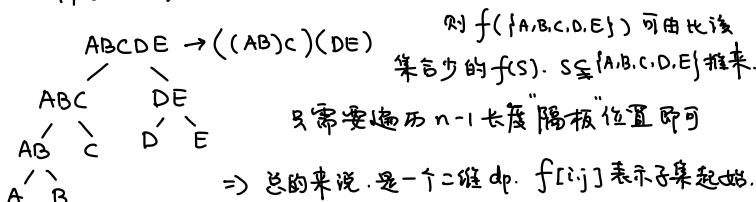
每踢一个点，才会计算 $(2S[i], f[i] + S[i])$ 与下一个旧点的斜率。
选最小值也一样，比较过斜率便踢掉，不会再比了。 $\mathcal{O}(n)$

6. Product of Sets

L_1, L_2, \dots, L_n 按顺序做笛卡尔积，求最小乘积数。

比如 $(L_1 \times L_2) \times L_3 = m_1m_2 + m_1m_2m_3$ ； $L_1 \times (L_2 \times L_3) = m_1m_3 + m_1m_2m_3$

⇒ 本质：是计算树的切分



• Time Complexity：朴素需要 $\mathcal{O}(n^3)$ 个状态，转移 $\mathcal{O}(n)$ ，则总 $\mathcal{O}(n^3)$

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

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

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

7. 最短路的 DP 视角

• Bellman-Ford (单源 $S \rightarrow$ 所有) $\text{dist}[k, u]$ 从 $S \rightarrow u$ 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)

$\text{dist}[k, u, v]$ ： u 到 v 的最短距离，只经过 u_1, u_2, \dots, u_k 中的点

提供了很好的转移性质：

$k-1 \rightarrow k$ 的转移只比较两个状态 不经 u_k 经过 u_k

不记： $\text{dist}[k-1, u, v]$

经过： $\text{dist}[k-1, u, v] + \text{dist}[k-1, v, v]$

算法：1. $\text{dist}[0, u, v] = d(u, v)$ for all $(u, v) \in E$. (otherwise ∞)

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

 for $u=1$ to $|V|$:

 for $v=1$ to $|V|$:

$\text{dist}[k, u, v] = \min \{ \text{dist}[k-1, u, v], \text{dist}[k-1, u, v_k] + \text{dist}[k-1, v_k, v] \}$

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

8. 树上 Max Independent Set

状态 $f(v)$ ：以 v 为根的子树
 $f(x) = \max \left\{ \frac{\sum f(c)}{\text{不选 } x}, 1 + \frac{\sum f(cc)}{\text{选 } x} \right\}$ 的最大独立集 size

• Time：有 n 个状态，而每个状态至多被检查 2 次：一次作儿子，一次作孙子
 \Rightarrow 总时间 $\mathcal{O}(n^2)$

• 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。 \Rightarrow 倒二一定不选，叶子一定全选

实现：一次 DFS：到叶子则选上（标记），退回把父亲标记。
 如果回溯后发现所有孩子标记 → 标绿 \Rightarrow 一次 DFS $\mathcal{O}(|V|)$ 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

设计一个向树靠拢 的算法

Cycle: 2

Clique: $n-1$

Tree: 1

⋮

分解成各 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)

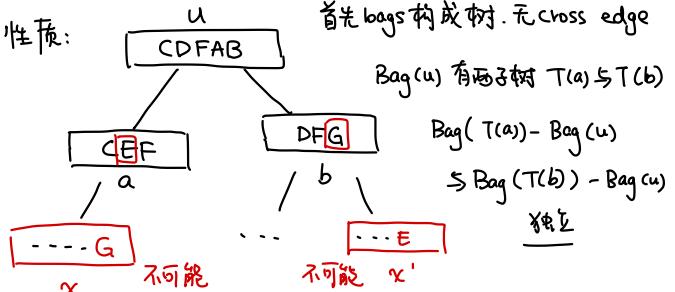
2. 存在 (u, v) 的边则 u, v 在同一 bag 中

$$\text{Treewidth} = \text{Maxsize(bag)} - 1.$$

(若 G 为树，则 Treewidth = 2 - 1 = 1.

即每条边设一个 bag 包含两侧的点)

△ 性质：



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： $f(S, u) =$

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

\downarrow
若有 c 已经有 S 中记数过
枚举在 $B(c)$ 中未被 S 决定是否选择的其余所有情况。

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: $f: E \rightarrow \mathbb{R}^+$, 定义域 $E \subseteq E$. (即将边映射到实数)

Capacity Constraint: $\forall e \in E, f(e) \leq C(e)$.

Flow Conservation: $\forall u \in V \setminus \{s, t\}, \sum_{(u,w) \in E} f((u,w)) = \sum_{(w,u) \in E} f((w,u))$ 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 $(u,v) \in E$, $C^f(u,v) = C(u,v) - f(u,v)$ 正向方向, E^f 中记录剩余多少 capacity
若 $(v,u) \in E$, $C^f(v,u) = f(v,u)$ 反向方向, E^f 中反向记录 flow 了多少

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

② While $\exists s \rightarrow t$ path on G^f :

find minimum capacity b edge on path. ← 每次更新路上最少处

for each $e = (u,v) \in \text{path}$:

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)

• Lemma 0: 流 $V(f) = f_{out}(L) - f_{in}(L)$

$$\begin{aligned} f_{out}(L) - f_{in}(L) &= \sum_{u \in L} f_{out}(u) - f_{in}(u) \leftarrow \text{相加成为 } L \text{ 的和.} \\ &= f_{out}(s) + \sum_{u \in L \setminus \{s\}} f_{out}(u) - f_{in}(u) \leftarrow \text{把 } S \text{ 情况单独} \\ &= V(f) + 0 \quad \text{由入=出约束为 } 0 \end{aligned}$$

• Lemma 1: For any flow f and cut $\{L, R\}$. $V(f) \leq C(L, R)$

$$V(f) = f_{out}(L) - f_{in}(L) \leq f_{out}(L) = \sum_{\substack{u \in L \\ \text{VER}}} f_{out}(u) \leq \sum_{(u,v) \in E} C(u,v) = C(L, R)$$

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

• Lemma 2: \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 $L \rightarrow R$ 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 $L \rightarrow R$ 反流.

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

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

\Rightarrow 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 \Rightarrow 得出最大 flow 后, 在 G^f 中可由 s 到达的边为 L , $R = V \setminus L$.

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

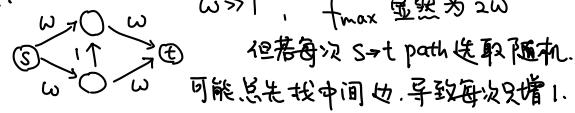
② Can halt → 并且这会和复杂度一起算.

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

① 边权 integer: 由于每次找到 $s \rightarrow t$ 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 (即每轮只增 1)

Bad Case:



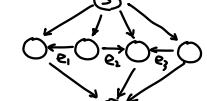
$w \gg 1$, f_{max} 显然为 $2w$

但若每次 $s \rightarrow t$ 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

$$e_1 = e_3 = 1, e_2 = \frac{\sqrt{5}-1}{2} (\text{满足 } e_2^2 = 1 - e_2)$$



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

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

BFS 关键: 在 G^f 中, 找的 $s \rightarrow t$ 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
 \Rightarrow 虽然 $dist(t)$ 只有 $1, 2, \dots, |V|, +\infty$ 共 $|V|+1$ 个取值, 但在每个值上有未知次数停顿.

\Rightarrow 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.

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

$\Rightarrow dist(u)$ 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

\Rightarrow 至多 $O(|E|)$ 条边. 每次循环有一条边 critical. 至多 $O(|V||E|)$ 循环

\Rightarrow 每次循环 BFS $O(|E|)$. 则总时间 $O(|V||E|^2)$

3. Dinic — 3D 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 $\rightarrow i+1$ 层的边
(即去除回边和层内边)

Algorithm:

① 在 LG 上找 block flow: 所有 $s \rightarrow t$ path 必须至少一条边 critical
也即将所有能到 $s \rightarrow t$ 的路都堵死

在 LG 上 DFS 拾到 $s \rightarrow t$ path: 找 critical 边并设置 flow.

找到 t 路 找到 critical 边

两种可能 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 $O(|V|)$
而一次至少删掉一条 critical 边 (到 t 或到 dead-end)

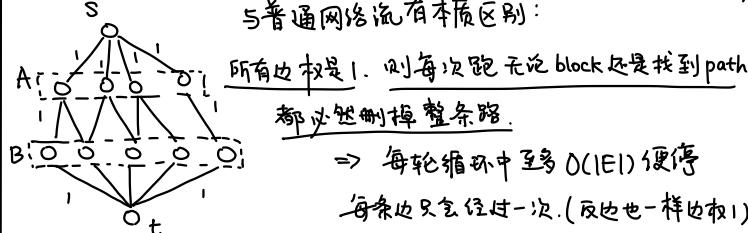
② 注意到一次循环后必从 LG 中 $k+1$ 层生出一条新边

找回 k 层:
或 在 $k+1$ 层内部

找回 k 层: $dist(t)$ 严格单增
在 $k+1$ 层内部: $dist(t)$ 增加 1

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 不重叠且点在二分图上 matching 就是配对问题) 与普通网络流有本质区别：



• Claim 1: 每次至多 $|E|$ 轮找全 block flow

• Claim 2: $\forall \{s, t\}$ 的入度与出度至少有一个是1

(Inductively, 二分图初始满足, \rightarrow 增广后 \leftarrow
入边与出边交换, 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)

构造 $s \rightarrow t$ 的网络流, 并设边权都是1. 找 edge disjoint

在最终停上 G^f 中看从 s 出了几条边

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)

• Claim 3: 至多 $2\sqrt{|V|}$ 轮循环. vertex disjoint = edge disjoint

—— 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

$\Rightarrow |V|$ 个点至多 $\sqrt{|V|}$ 个点不相交路

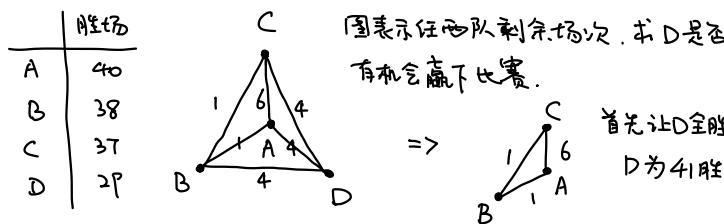
$\Rightarrow G^f$ 此后最大流至多 $\sqrt{|V|}$

$\Rightarrow \sqrt{|V|}$ 轮可停止.

\Rightarrow 每轮循环 $O(|E|)$, 共 $O(\sqrt{|V|})$ 轮

\Rightarrow Hopcroft-Karp-Karzanov (二分图上跑 Dinic) $O(|E|\sqrt{|V|})$

5. 网络流应用2：锦标赛胜场



\Rightarrow A至多 win 1, B至多 win 3, C至多 win 4

构建网络流

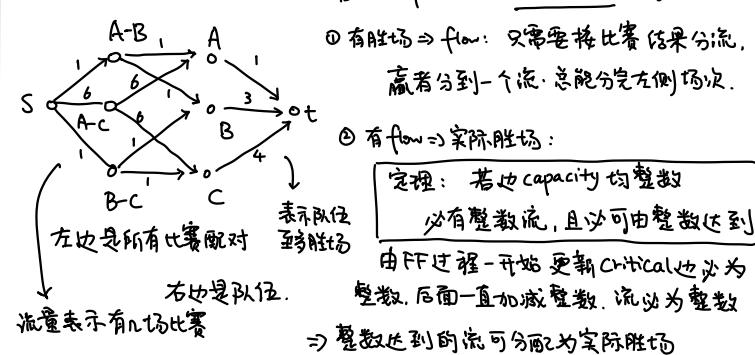
若 max-flow 能让左侧流满, 则存在分配

① 有胜场 \Rightarrow flow: 只需将比赛结果分流, 赢者分到一个流, 总能分完左侧场次.

② 有 flow \Rightarrow 实际胜场:

定理: 若边 capacity 均整数

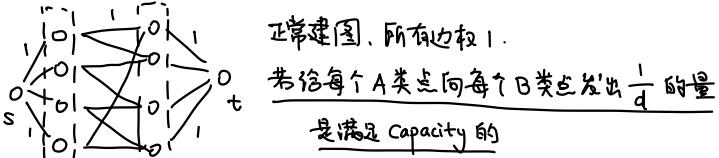
必有整数流, 且必可由整数达到



6. 定理1: regular 二分图必有 perfect matching

\downarrow
(\forall vertex, same degree) (\forall vertices matched)

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



\Rightarrow flow 可达 $|A|$. 且这是一个割大小, 就是最大流

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

7. 定理2: Hall's Marriage:

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

① " \Leftarrow " 显然. 否则 $\exists S, |S| > |N(S)|$. 那么 S 个人必不可配上.

② " \Rightarrow " 构造网络, 该网络满足右侧性质.

欲证有 $A \rightarrow B$ 全映射, 要找到 $|A|$ 的网络流

\Rightarrow 证明该网络最小割 = $|A|$

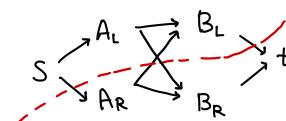
Proof: 将割分成三类

中间边权设为1也行
 ∞ 便于证明 \rightarrow

1. 直接切 ⑤. 割 = $|A|$

2. 直接切 ④. 割 = $|N(A)| > |A|$

3. 若割如下所示. 部分割 ⑤. 部分割 ④. 分 $\{S, A_L, B_L\} \{A_R, B_R, t\}$ 两组



严格说这样切是割: $t \rightarrow |A_R|, t \rightarrow |A_L| \rightarrow |B_R|$ 但若割中有 $|A_L| \rightarrow |B_R|$ 直接 ∞
 \Rightarrow 故正常割的图中不存在 $A_L \rightarrow B_R$ 也

故切掉的边只剩两侧:

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

\Rightarrow 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

$$\begin{aligned} \max \quad & C_1x_1 + C_2x_2 + \dots + C_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1 \\ & \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m \\ & x_1, \dots, x_n \geq 0 \end{aligned}$$

性质：

- ① always \exists optimum (x_1, \dots, 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)
仍能 bound 在多项式 (run fast in practice) (椭球法、内点法、均理论 bound 多项式)

3. 标准形

- ① max ② 约束 \leq 若原 = 则松弛弛 \geq 则 $x \rightarrow -x$
- ③ $x_i \geq 0$ 约束，换成 $x_i - x_i^+$, $x_i^+, x_i^- \geq 0$

4. 网络流 \rightarrow LP (Ford-Fulkerson 就是 simplex 算法，只不过选方向很聪明)

变量是 f_{uv} for all $(u,v) \in E$

$$\begin{aligned} \max \quad & \sum_{(s,u) \in E} f_{su} \\ \text{s.t.} \quad & 0 \leq f_{uv} \leq C_{uv}, \forall (u,v) \in E \\ & \sum_{(v,u) \in E} f_{vu} = \sum_{(u,w) \in E} f_{uw}, \forall u \in V \setminus \{s,t\} \end{aligned}$$

5. Duality 理解

$$\begin{aligned} \max \quad & x_1 + 6x_2 + 13x_3 \\ \text{s.t.} \quad & x_1 \leq 200 \\ & x_2 \leq 300 \\ & x_1 + x_2 + x_3 \leq 400 \\ & x_2 + 3x_3 \leq 600 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

希望求出一个最小的值

$$\begin{aligned} & x_1 + 6x_2 + 13x_3 \\ & \leq (y_1 + y_3)x_1 + (y_2 + y_3 + y_4)x_2 + (y_3 + 3y_4)x_3 \\ & \leq 200y_1 + 300y_2 + 400y_3 + 600y_4 \leftarrow \\ & \text{由于 } x_1, x_2, x_3 \geq 0, \text{ 目标变为最小化这个值} \\ & \text{则要求 } y_1 + y_3 \geq 1, y_2 + y_3 + y_4 \geq 6, y_3 + 3y_4 \geq 13 \end{aligned}$$

$$\begin{aligned} \max \quad & C^T x \\ \text{s.t.} \quad & Ax \leq b \Leftrightarrow \min \quad b^T y \\ & x \geq 0 \quad \text{强对偶声明 LP 问题} \\ & y \geq 0 \quad \text{prim OPT} = \text{dual OPT.} \end{aligned}$$

6 LP-relaxation 将 Integer LP \rightarrow LP (ILP 为 $\exists x \in \mathbb{Z}^n$)
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 relaxes ILP \rightarrow LP 为：

$$\begin{aligned} \min \quad & \sum_{v \in V} x_v \quad (\text{本身是 } x_v \in \{0,1\} \text{ 代表选/不选}) \\ \text{s.t.} \quad & x_u + x_v \geq 1 \quad (u,v) \in E \\ & 0 \leq x_v \leq 1 \quad v \in V \end{aligned}$$

算法结果对每条边会有分数的预测

0.3
0.7

\Rightarrow 一个近似处理：把分数中 $\geq \frac{1}{2}$ 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. $\text{OPT(ILP)} \geq \text{OPT(LP)}$

$$\begin{aligned} \text{而 } \text{OPT}(LP) = \sum_{v \in V} x_v = \sum_{x_v < \frac{1}{2}} + \sum_{x_v \geq \frac{1}{2}} \geq 0 + \sum_{x_v \geq \frac{1}{2}} \frac{1}{2} \\ = ALG \cdot \frac{1}{2} \quad \text{数量即我们选的点数} \end{aligned}$$

$$\Rightarrow ALG \leq 2\text{OPT}(LP) \leq 2\text{OPT(ILP)}$$

• Vertex Cover \leftrightarrow Matching (König-Egervary 定理)

Dual:

$$\begin{aligned} \max \quad & \sum_{e \in E} x_e \\ \text{s.t.} \quad & \sum_{(u,v) \in E} x_{uv} \leq 1, \forall v \in V \\ & x_{uv} \geq 0 \end{aligned}$$

(相当于每个点被选择量不能超过 1 次。
match 了不能再 match.)

7. Max Flow \leftrightarrow MinCut 用对偶证明该定理

$$\begin{aligned} \min \quad & \sum_{(u,v) \in E} y_{uv} c_{uv} + 0, (u,v) \in E \\ \text{s.t.} \quad & \sum_{u \in S} y_{su} + \sum_{u \in T} y_{ut} \geq 1, (s, u, t) \in E \\ & y_{ut} - y_{su} \geq 0, (u, t) \in E \\ & y_{uv} - y_{su} + y_{ut} \geq 0, (u, v) \in E, u \in S, v \in T \\ & y_{uu} \geq 0, (u, u) \in E \end{aligned}$$

(这里 y_{uv} 的意义是 (u, v) 是否为最小割一部分
 y_{ut} 的意义是最小割中 u 在 S 是 + 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. 证明其中一个问题是整数解 (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, 只能说明 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 的来 (比如是整数 match, 但有的边负, 有的正)

定理：If $A \in \mathbb{R}^{m \times n}$ is total unimodular ($\det(\text{任一子式}) = 0, \pm 1$)
且 b 是全整数向量 \Rightarrow polytope $P = \{x : Ax \leq b\}$ 都是 integer vertices
 \Rightarrow 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: $x_i = \frac{\det(A'_i | b')}{\det(A'_i)}$ \rightarrow 为 ± 1 而 b' 也整数 $\Rightarrow x_i$ 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

左侧边的变量部分必为 I
约束矩阵:

$ E $	$ V -2$
-------	---------

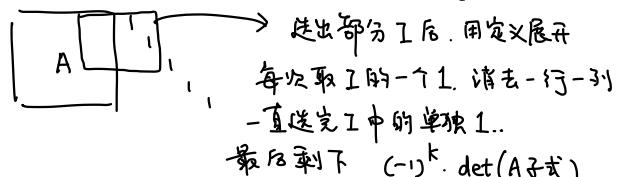
1	1
1	-1
1	1
1	-1

因为约束中每条边对应且仅
对应 f_{uv} .
右侧, 每行要么 1 个数要么 2 个数

Lemma 1: 单位阵 total unimodular

Lemma 2: A 是 total unimodular $\Leftrightarrow A^T, [A|I], [I|A]$ 也是

3 步:



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.

$$\det(A_{k+1}) = \pm 1 \times \det(A_k) \in \{-1, 0, 1\} \quad \checkmark$$

若整个子阵不存在 ①, 即每行都有一个 1 和 -1.

则 $\boxed{\text{把所有列加到第 3 列}}$, \det 不变, 而第一列变成了全 0

\Rightarrow 此情况下 $\det(A_{k+1}) = 0$. \Rightarrow 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 $x(-1)$ 加所有 B 类的列 \Rightarrow 全 0) $\text{Mix}|E|$

$$\begin{array}{cccccc} A_1 & e_1 & e_2 & e_3 & e_4 \\ \hline B_1 & 1 & 1 & 1 & 1 \\ B_2 & 1 & 1 & 1 & 1 \\ \vdots & & & & & \end{array}$$

- 于是由 lemma 3 和定理，这个 LP 可由整数解到
下一步要从给定的整数解中恢复出 max flow 的边。

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

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

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} 表示是否割一部分，则左侧加 C_{uv} . 右侧也加 C_{uv} .

$$C(L, R) = \sum_{\text{某割}} C_{uv} \leq \sum_{\text{某割}} y_{uv}^* C_{uv} \leq \text{OPT(dual)} \leq C(L, R)$$

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

8. Zero-Sum Game

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

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 \nearrow 的 gain 或 B 在 \searrow 的 loss.

$\{3, j\}$: 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 $x = \{x_1, \dots, x_m\}$ B 的 $y = \{y_1, \dots, y_n\}$

$$U_A(x, y) = x^T G y \quad U_B(x, y) = -x^T G y = -\sum G_{ij} x_i y_j$$

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 : $\max_x \min_y \sum G_{ij} x_i y_j$

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

Lemma 1: 最优解为 pure Strategy

因为展开 $\max_x \min_y \sum G_{ij} x_i y_j = y_1 \sum G_{i1} x_i + y_2 \sum G_{i2} x_i + \dots + y_n \sum G_{in} x_i$

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

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

写成 LP 形式： $\max_x z$

$$\text{s.t. } \sum G_{ij} x_i - z \geq 0 \quad \forall j = 1, \dots, n$$

$$x_1 + \dots + x_m = 1 \quad (\text{prob distribution})$$

$$x_1, \dots, x_m \geq 0$$

Dual: \min_w

$$\text{s.t. } \sum G_{ij} y_j \leq w \quad \forall i = 1, \dots, m$$

$$y_1 + \dots + y_n = 1$$

$$y_1, \dots, y_n \geq 0$$

再由 Strong duality



$$\text{Minimax Theorem: } \max_x \min_y \sum G_{ij} x_i y_j = \min_y \max_x \sum G_{ij} x_i y_j$$

⑤ $SS \leq_k Partition$: $S = \{s_1, \dots, s_n\}$, 判断是否有分隔使 $\sum_{a \in A} a = \sum_{b \in B} b$
 $\leq_k Partition^+$: 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 $2k - \sum_{i=1}^n a_i$

$$SS(\{a_1, \dots, a_n\}, k) \Rightarrow Partition(\{a_1, \dots, a_n, 2k - \text{sum}\})$$

则若前者为 Yes, 有 $\sum a' = k$, 则后者必有分割 $\sum a' = k$, 剩余还是 k
 则后者有一个分割, 不妨设将 a_1, \dots, a_n 分为了 $X \subseteq S$.

则无论 $X + (2k - \text{sum}) = \text{sum} - X$ 还是 $X = \text{sum} - X + (2k - \text{sum})$, 均有一部分为 k .

• 对于 Partition⁺, 可由 Partition reduce 过来.

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

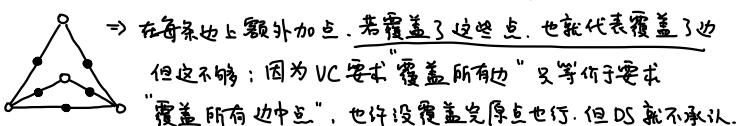
若 $\{s_1, \dots, s_n\}$ 有分割, 必可写成 $P_1 - n_1 = P_2 - n_2$ (positive & negative)

则在绝对值中, 必有另一分割 $P_1 + n_1 = P_2 + n_2$.

反之绝对值有分割原数组也必有 (本质是正数与负数部分各自差值相同)

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

类似 Vertex Cover 覆盖边. 希望转化为覆盖点 (注意 VC 是覆盖边)



\Rightarrow 将原点补成完全图. 保证了原点均被覆盖

• 有 VC 的 Yes \Rightarrow DS 的 Yes : VC 首先 cover 所有边 \Rightarrow cover 所有新加点
 \downarrow 由完全图, 也必 cover 所有原点

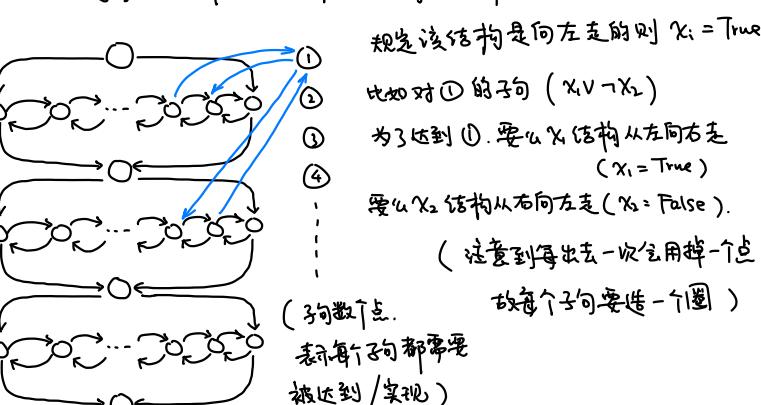
• DS 的 Yes \Rightarrow VC 的 Yes : 把所有连了的新加点任意换成两端的某个原点
 新加点换成原点, cover 的边数一定不降.

⑦ SAT \leq_k 有向图 Hamilton Path 存在性.

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

给每个变量构造一个结构

性质 1. 从 s 起始走完的唯一方法
 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.

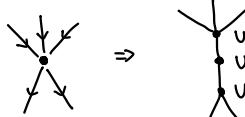


• SAT 的 Yes \Rightarrow 有 HP: 依照赋值决定每个圈左/右走, 由于保证了每个 clause True, 则必有其中一个值为 True, 则从这个变量的圈可达该子句.

\Rightarrow 所有子句可达 \Rightarrow Hamilton Path

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

⑧ 有向 HP \leq_k 无向 HP



把一个点拆成三个. \Rightarrow 避免了从原图中的入边进入边出的情况

(1) 有向 HP \Rightarrow 无向 HP. 直接走对应边即可
 (2) 无向 HP \Rightarrow 有向 HP.

性质: U_{in} 必接上一个的 U_{out} . U_{out} 必接 U_{in}
 由性质, 若无向 HP 有一组点顺序为 $U_{in} - U_{mid} - U_{out}$ 或 $U_{out} - U_{mid} - U_{in}$.

则所有组顺序都是这个 \Rightarrow 对应有向图中正路或反路. \Rightarrow 也以存在

⑨ 无向 HP \leq_k Hamilton Cycle.

大致: $\boxed{\text{图上} \rightarrow \text{Independent Set}; \text{数值相关} \rightarrow \text{Subset Sum}; \text{否则} \rightarrow \text{SAT}}$

Approximate Algorithm (假设是 minimize 问题)

1. ratio 定值 $ALG(\text{Input}) \leq \alpha \cdot OPT(\text{Input}) \quad \forall \text{ Input.}$

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

$n=100$ 时 $ALG \leq \log 100 \cdot OPT$

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

3. PTAS & FPTAS (Polynomial Time Approximation Scheme)
 多项式时间逼近任意常数近似比

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

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

4. FPT (Fixed-Parameter Tractable)

精确求 OPT. 但不一定多项式.

$O(f(k) \cdot n^c)$ \downarrow 卡是一个描述输入情况有多差的参数.

\downarrow 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”