第I Strongly Connected Components

1. Basic Concepts.

* 有向图: $G=\langle V,E\rangle$. 边集中的每条边都是有向的.
* neighbor: 同一条边上的端点.
* incidence: 边和该边上的顶点.
* 有向图中顶点的 degree: 入度. 出度.

主要问题:

1. reachability (从A到B?) => st-CON
   given: directed graph $G=\langle V,E\rangle$; $s,t\in V$.
   return: True if t reachable from s in G, else False.

2. Strongly connectivity. (所有顶点均互相可达?) => STRONG CONNECTIVITY.
   given: directed graph $G=\langle V,E\rangle$.
   return: True if strongly connected else False.

3. Cyclicity:
   given: directed graph $G=\langle V,E\rangle$.
   return: True if G is cyclic (has a cycle) else False.

等价:

1. For reachability: 用 DFS 找路径.

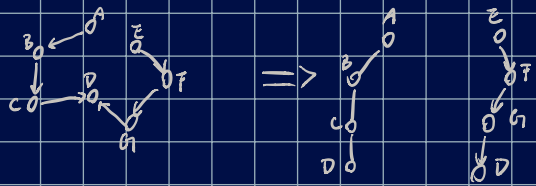
2. For strongly connectivity:
   用 Kosaraju 算法, Tarjan.

   Kosaraju: DFS + Edge Reversal + DFS, 时间复杂度: $O(|E|+|V|)$

   Tarjan: 基于DFS遍历, 对每个点维护 low: 首次发现时间和 low: 强连通分量标记
   由于有向图中的强连通分量必会被遍历到2次: 在DFS返回时将同一个强连通分量中的所有元素标记. 时间复杂度: $O(|E|+|V|)$

   注: 和 Kosaraju 不同: Tarjan算法找到的是给定点v所属的强连通分量,
   而 Kosaraju 只可用于检查全图的强连通性.

3. For Cyclicity:
   用 Topological Sort. 基于定理: 只有无环(有向)图才有拓扑排序.
   拓扑排序: 得到无环图的"排序树".
   从 0入度的点开始作为根节点, 将其所有子节点入度-1后类似DFS继续遍历,
   直到再也找不到 入度为0 的点为止.



> A topological sort(ing) of a directed graph $G=(V,E)$ is an ordering of its vertices as $V=\{u_0,\dots,u_{n-1}\}$ such that, for all edges (u_i,u_j) we have $i<j$.

反知: 若图中有环, 则由于该环内所有点入度均不为0, 故其必无拓扑排序.
同时有: 时间复杂度为 $O(|E|+|V|)$.

第11 Union-Find

无向图：$G=\langle V,E\rangle$，边无向，不再关注"强连通性"，只需考虑连通性即可。
同时不再有入度、出度、只有度（因为边无向了）

主要问题：

1. undirected st-CON：无向的 reachability 问题。
   given：无向图 $G=\langle V,E\rangle$，$s,t\in V$.
   return：True iff t reachable from s in G.

2. connectivity：更弱(一般)的图连通性。
   given：无向图 $G=\langle V,E\rangle$.
   return：True iff G connected else false.

3. connected components："连通子图"
   given：无向图 $G=\langle V,E\rangle$
   return：该图的所有连通子图。

下面主要关心：连通子图问题：
1. 可用 simple DFS 在 linear time 中解决。
2. 可利用 union-find 结构。

Union-Find 相关定义：
1. Partition（划分）：

- A partition of V is a collection of sets $\{A_1,\ldots,A_m\}$ such that:
  - $A_1\cup\cdots\cup A_m=V$;
  - $A_i\cap A_j=\emptyset$ for all i,j, $(1\le i<j\le m)$;
  - $A_i\neq\emptyset$ forall i $(1\le i\le m)$.
- We refer to the elements of a partition (sets of vertices) as cells. A_1, \cdots, A_m

2. Basic Operations：
   1. makeset. ~ 为给定的 vertex 赋予一个划分。
   2. union. ~ 合并两个 partition cell
   3. find. ~ 找到给定 vertex 所属的 partition.

3. 关于 union-find 结构的优化：
   1. 在作 union 时，总 update smaller sets.（less operations）即：将 size 更小的 tree 的 root 指向大树的 root 上。
   2. Path Compression：flatten the tree, improve lookup speed.

4. 相关 Thms：

Lemma
In a series of operations of `makeSet`, `union` *and* `find` *on n elements using the size-heuristic, no element can have its* `cell` *field assigned more than $\lfloor\log n\rfloor+1$ times.*

Proof.
Whenever $v\to$`cell` changes, the cardinality of $v\to$`cell` at least doubles. But $1\le|v\to\texttt{cell}|\le n$. This can happen no more than $\lfloor\log n\rfloor$ times. □

Theorem
With the above implementation, the running time of `union-find`(G) *is $O(m+n\log n)$, where $n=|V|$ and $m=|E|$.* (time on updating cells)

Proof.
The total number of calls to `makeSet`, `union` and `find` is linear in $m+n$. Each call involves a constant amount of work, apart from updating $v\to$`cell`. But each of the n vertices can have its cell-pointer updated in this way at most $\lfloor\log n\rfloor$ times, yielding total work $O(m+n+n\log n)=O(m+n\log n)$. □

(For non-optimized Union-Find)

第三章 Fast & Slow

Basic Notations:

1. Polynomial:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

Polynomial-Bounded:

$\forall n.\ f(n) \le p(n) \Rightarrow f(n)$ is polynomial bounded.

2. Exponentially Bounded:

$\forall n.\ f(n) \le 2^{p(n)}$

3. Double-Exponentially Bounded:

$\forall n.\ f(n) \le 2^{2^{p(n)}}$

4. k-Tuple exponentially Bounded:

$\forall n.\ f(n) \le 2^{2^{\cdots^{2^{p(n)}}}}$ (# = k)

5. Tower (tetration) function:

$$t(1) = 2 \quad t(2) = 2^2 \quad t(3) = 2^{2^2} \quad t(4) = 2^{2^{2^2}} \quad \cdots \Rightarrow t(n) = 2^{\cdot^{\cdot^{\cdot^{2}}}} \} n\text{个}2.$$

一些基础的时间复杂度类：

1. Basis: 定义抽象的 "Loop Language" 描述算法

The language, LOOP has the following constructs:

```
x= y
x= 0
x++
return x
loop (x) { ··· }
```

```
y1= x
loop(y1){x++}
return x
```

返回值就是2x

```
y2= x
x= 1
loop(y2){
   y1= x
   loop(y1){x++}
   }
return x
```

```
y2= x
x= 1
loop(y2){x= 2x}
return x
```

```
y3= x
x= 1
loop(y3){
   y2= x
   x= 1
   loop(y2){
      y1= x
      loop(y1){x++}
      }
   }
return x
```

```
y3= x
x= 1
loop(y3){x= 2^x}
return x
```

返回值是 t(x)

2. 基于 Loop 语言定义的复杂度类：$\mathcal{L}$.

$\mathcal{L}_k$：所有可由最多 k 层嵌套的 Loop language 所定义的 $\mathbb{N} \to \mathbb{N}$ 函数类.

且有：$\mathcal{L}_1 \subsetneq \mathcal{L}_2 \subsetneq \cdots \subsetneq \mathcal{L}_4 \subseteq \cdots$

3. 另一种复杂度类：Grzegorczyk hierarchy.

* $\mathcal{E}_1 \subsetneq \mathcal{E}_2 \subsetneq \cdots \subsetneq \mathcal{E}_4 \subsetneq \cdots$
* 对 $k \ge 3$：$\mathcal{E}_k = \mathcal{L}_{k-1}$.

进一步引入的一些定义：

1. Elementary:

any function in $\mathcal{L}_2$. 即：bounded by $2^{p(n)}$.

2. Primitive Recursive:

即所有在 $\mathcal{L}$ hierarchy or $\mathcal{E}$: Grzegorczyk hierarchy 的函数.

$$\Rightarrow \bigcup_{i=1}^{\infty} \mathcal{L}_i = \bigcup_{i=1}^{\infty} \mathcal{E}_i.$$

4. 增长更加迅猛的函数 : Ackermann Function

Definition:

- The Ackermann function :

$$A(m,n) = \begin{cases} n+1 & \text{if } m=0 \\ A(m-1,1) & \text{if } m>0 \text{ and } n=0 \\ A(m-1, A(m, n-1)) & \text{otherwise.} \end{cases}$$

注: Tower Function `t(x)`, 对应L(3), 位于Grzegorczyk Hierarchy 的底端 (\epsilon 2),
而Ackermann Function增长的速度如此的快, 以至于它压根不在 Grzegorczyk Hierarchy 中

- Notation: $A^{(k)}(x) = \overbrace{A(A(\cdots A(x)\cdots))}^{k \text{ times}}$
- An Ackermann-like function:

$$A_0(x) = x+1$$
$$A_{i+1}(x) = A_i^{(x)}(x).$$ repeat A_i(x) for x times

$$A(x) = A_x(2).$$

Ackermann Function 的反函数: $\alpha(n)$.

$$\alpha(n) = \min\{i \geq 0 : A(i) \geq n\}.$$

the notation means: as the input A(n) increases, eventually its output will be bigger than n

Note: $\alpha(A(n)) = n$.

下面使用上述铺垫证明: Optimized Union-Find is almost linear.

Theorem \alpha(n): essentially no bigger than 4

Using the tree implementation, the algorithm `union-find`$(G=(V,E))$ *runs in time $O((n+m)\alpha(n))$, where $n=|V|$ and $m=|E|$.*

Proof: introduce amortized analysis to analyse the running time of sequence with m union and find operations. on the partition initially consisting n distint single-element sets.

1. U: a tree defined by all union operations without having performed any path compressions.
   define rank of node as:

$$r(v) = \lfloor \log(n(v)) \rfloor + 2.$$

   then we got: for $\forall v$, $n(v) \geq 2^{r(v)-2}$; $\forall v$, $r(v) \leq \lfloor \log n \rfloor + 2$.

   Lemma 1: if node w is v's parent, $r(v) < r(w)$

   Lemma 2: for given number s, # of nodes with rank s is at most $\frac{n}{2^{s-2}}$.

2. Now analysis the time takes to perform the m union and find operations.
   i) since each `union` operation take $O(1)$, in all can perform at most n-1 unions:
   $\Rightarrow$ cost for all union operations is $O(n)$.

   ii) analyse path compression: consider how path compression affects performance of 'find' operation.
   because: 1) for each node v, its parent node $p(v)$ may be changed.
   2) when changing its parent node, the value of $r(p(v))$ will also increase.

   Define a labeling function $L(v)$ for each v:
   * in each step 't', $L(v)$ is defined as:
   " the largest i, so that $r(p(v)) \geq A_i(r(v))$.

   then can get: for all nodes 'v' and step 't': $L(v) < \alpha(n)$ (*)

   then analyze the computational task in 'find' operation:
   1) if v has an ancestor w in the path P such that $L(v) = L(w)$: cost 1 for v itself.
   2) if v doesn't have such ancestor: cost 1 for 'find' operation.

   Then we can see:
   1) cost for any 'find' operation is bounded by the number of distinct $L(v)$ values in P:
   (in worst case every distinct $L(v)$ value will cost 1 to 'find' operation)

⇒ total charge to all 'find' is: $O(m \cdot \alpha(n))$. (*)

2) consider the cost assigned to each vertex v:

by definition: in this case v has an ancestor, s.t. $L(v) = L(w) := i$.

⇒ in this time 'c' we have:

$$\begin{cases} r(p(v)) \geq A_i(r(v)) \\ r(p(w)) \geq A_i(r(w)) \end{cases}$$

suppose: particularly $\exists k \geq 1$, s.t. $r(p(v)) \geq A_i^{(k)}(r(v))$

then: $r(z) \geq r(p(w)) \geq A_i(r(w)) \geq A_i(r(p(v))) \geq A_i(A_i^{(k)}(r(v))) \geq A_i^{(k+1)}(r(v))$.

(z is the root of path p)

since path-compression, at next time 't+1': z becomes v's parent, we got $r(p(v)) = r(z) \geq A_i^{(k+1)}(r(v))$.

implies: at most $r(v)$ costs assigned on vertex v. ☆ $L(v) = i+1$

since after costs are charged, $L(v)$ increases at least 1:

since $L(v) < \alpha(n)$, $L(v)$ can increase at most $\alpha(n)-1$ times:

the total cost charged on each vertex 'v' is:

$$r(v) \cdot \alpha(n) \leq s \cdot \alpha(n) \cdot \frac{n}{2^{s-2}} = n \cdot \alpha(n) \cdot \frac{s}{2^{s-2}}, \text{ denote } r(v) = s, \text{ from lemma 2.}$$

finally sum over all possible rank values:

$$\sum_{s=0}^{\lfloor \log n \rfloor + 2} n\alpha(n) \frac{s}{2^{s-2}} \leq \sum_{s=0}^{\infty} n\alpha(n) \cdot \frac{s}{2^{s-2}} = n\alpha(n) \cdot \sum_{s=0}^{\infty} \frac{s}{2^{s-2}} \leq 8 \cdot n\alpha(n)$$

since $O(m\alpha(n) + n\alpha(n)) = O((m+n)\alpha(n))$:

total charges made to all 'union' and 'find' is $O((m+n)\alpha(n))$.

§IV. Searching Strings.

目标：介绍教科书用于在大string中搜索子string的算法.

1. String Matching Problem

问题：在给定 instance 中找到目标子 string 的首个 instance？

⇒ 构建对问题的形式化描述：

Σ: alphabet, finite, non-empty set.

T: main string (主串) defined on Σ.

P: pattern string (模式串) defined on Σ.

目标：在 T 上找到 P 的首个匹配的位置.

教科书问题：MATCHING.

given: string T, P over alphabet : Σ.

return: index i, such that P first occurs in T at position i, or "no match".

2. naive algorithm:

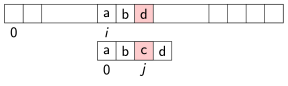
对主串 T 上的每个位置均检查一下是否为 P 的一个 occurrence.

基本思想：从字符串的每个位置开始遍历检查是否存在对应的子字符串

- Here is a naive algorithm

```
begin naiveMatch(T,P)
    for i = 0 to |T| − |P|
        j ← 0
        until j = |P| or T[i + j] ≠ P[j]
            j++
        if j = |P|
            return i
    return "No match"
end
```

- Graphically



- Running time is $O(|T| \cdot |P|)$.

时间复杂度：$O(|T| \cdot |P|)$.

3. Rabin-Karp 算法：

利用 Hashset 中的"杂数编码"对模式串和主串编码并尝试匹配.

记 $|\Sigma| = b$, 则 alphabet 中的每个字母均可用一个数作一编码.

则有：1. $P := P[0] \cdot b^{m-1} + P[1] \cdot b^{m-2} + \cdots + P[m]$.

2. $T := T[0] \cdot b^{n-1} + T[1] \cdot b^{n-2} + \cdots + P[n]$. (记 $|P| = m$, $|T| = n$).

亦有：$(T[i : i+m-1] - T[i] \cdot b^{m-1}) \cdot b + T[i+m] = T[i+1 : i+m]$.

故对 $\forall i = 0, 1, \cdots, n-m$, 匹配：$T[i : i+m-1] \equiv P \pmod q$?

若 $T[i : i+m-1] \equiv p \pmod q$, 则进一步 check：是否 $T[i : i+m-1] = P$.

因"两个不同数模 q 的值相同"的概率为 $1/q$

⇒ worst case: $O(|T| \cdot |P|)$. expected performance: $O(\underbrace{n+m}_{\text{算 } t \text{ 算 } p} + \underbrace{m \cdot (\frac{n}{q})}_{\text{对给定的值, } T \text{ 中最多有多少处引起的 collision}}) \approx O(m+n)$. (一般字母表大于 m)

Here is the algorithm

```
begin Rabin-Karp(T,P,q,b)
    m ← |P|
    t ← T[0] · b^(m−1) + ··· + T[m − 1] · b^0  mod q
    p ← P[0] · b^(m−1) + ··· + P[m − 1] · b^0  mod q
    i → 0
    while i ≤ |T| − m
        if p = t
            j ← 0
            while P[j] = T[i + j] and j < |P|
                j++
            if j = |P|
                return i
shift T     t ← (t − T[i] · b^(m−1)) · b + T[i + m]  mod q
        i++
    return "No match"
end
```

4. KMP算法.

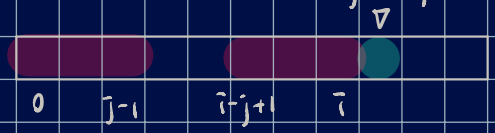
原理：无效字符串比较的多余量 ⇒ 设法减少比较的次数.

利用上一次匹配失败所含的信息：跳过不可能成功的字符串比较.

目的：利用"next"数组，在某次匹配失败后，利用offset跳过绝无可能匹配成功的位置.

next数组 $\pi(i)$ 定义为：

* i ~表示为原子串：$P[0:i]$. (对应在matching中，在i处失配的情况)
* $\pi(i)$ 的值 ~假设记 $\pi(i)=j$，则表示：$\pi[0:j-1] = \pi[i-j+1:i]$.



故：若在某处匹配时，恰在i处失配，则可直接从模式串的j处，即$\pi(i)$处开始继续匹配.

```
begin KMP(T, P)
    compute-π(P)        这一步就是计算的新位置k
    i ← 0, j ← 0
    while i < |T|
        if P[j] = T[i]  如果当前指向的字符都相等
            if j = |P| − 1      检查完了, 完全匹配
                return i − |P| + 1
            else
                i++, j++        全部递增继续检查
        else if j > 0   在一个合法的位置发现不匹配, 直接从next矩阵 (pi) 里找到需要跳转到的下一个位置
            j ← π[j]
        else
            i++
    return "Not found"
end
```

计算next数组 $\pi()$：

5.7 Flow Network.

基本定义:

网络流问题基于有向图. 有向图 $G=\langle V,E\rangle$ 中, 每条边有一个表示 capacity 的权值. 且 网络流中定义了源点 s 和汇点 t. 记为: (V,E,s,t,c). $c: E \to \mathbb{N}$. 给边赋权值的函数.

流网络 $N=(V,E,s,t,c)$ 中的一个流 (flow) 表为函数 $f: E \to \mathbb{R}^+$, (即: 为每条边赋流量).

满足:

1. 每条弧上 流量 $\leq$ 容量. 注: 对每个图中的节点 v 言: 它的 positive flow 和 negative flow 值是相同的, 显而易见!
2. 除源点 s 和汇点 t: 其余任何点的流入流量和流出流量和为 0.

称其为 可行流. 关注问题: 给定的流网络的 最大可行流?

> 问题: 给定某个流网络, 确定它的最大流
>
> **MAX FLOW**
> Given: A flow network N
> Return: An optimal flow f for N.

求给定流网络的最大流: Ford-Fulkerson 算法.

辅助定义:

1. auxiliary directed graph:

$N_f := \langle V, E_f\rangle$, $E_f := \{(u,v)\in E \mid f(u,v) < c(u,v)\} \cup \{(u,v) \mid (v,u)\in E,\ f(v,u)>0\}$.

i.e. 增广图中所有弧: either 流量 < 容量, or 作为反向弧, 流量 > 0.

$\Rightarrow$ 增广图上的所有前向弧均可继续添加流量

且 所有反向弧上的流量均可继续减少.

注意什么是"反向弧":

若弧 $(v,u)\in E$, 且 $f(v,u)>0$. 则称 (u,v) 这条 在残余图中的反向为 (v,u) 的反向弧.

$\Rightarrow$ 给定流量图中任何边若其流量 >0, 则构造 augmented graph 中有一条反向弧.

- Let $N = (V, E, s, t, c)$ be a flow network and f a flow for N.
- The auxiliary directed graph N_f, is $G = (V, E_f)$, where E_f is the set of pairs:

$$\{(u,v) \in E \mid f(u,v) < c(u,v)\} \cup \{(u,v) \mid (v,u) \in E \text{ and } f(v,u) > 0\}.$$

2. auxiliary directed path:

即增广图中的路径. 注: 增广路径中, 弧的方向不一定要和有向图中的方向一致 (因为可以包括反向弧)

最大流-最小割 Thm:

若给定流网络 (V,E,s,t,c) 中的一个可行流 f, 则:

f 为 optimal flow, 当且 该流 f 的增广图 N_f 中不存在从 s 到 t 的增广路径

$\Rightarrow$ Ford-Fulkerson 算法的思路: 不断在 f 的增广图 N_f 中寻找和添加 从 s 到 t 的增广路径

算法:

- The Ford-Fulkerson algorithm:

```
begin flowMax(V,E,s,t,c)
    set f(e) = 0 for all e ∈ E          start from the zero flow
    while reachable(N_f,s,t)            build auxiliary graph and check its edges
        let e_1, ..., e_m be the edges on a path from s to t
        for 1 ≤ i ≤ m
            if e_i ∈ E, increment f(e_i)        increase from the positive edge
            else decrement f(e_i^{-1})          decrease from the negative edge
                                                => increase overall flow
    return f
```

有次改变, 1 条路径 (即: 1)

因为增广图中的 e_i 是流网络图中也的反向.

1. Ford-Fulkerson starts with empty flow then augments
   $\Rightarrow$ optimal flow gained is integral. (整数流)
2. Ford-Fulkerson 适用于 rational-valued edge capacities:
   总可用 common denominator 将其全变为 integer.

Theorem

If $N = (V, E, s, t, c)$ is a flow network with integral capacity-function c, then `flowMax(V,E,s,t,c)` *returns an optimal flow for N.*

Corollary

If a flow network N has integral capacities, then there exists an optimal flow for N; in fact, there exists an optimal flow for N which is integer-valued.

时间复杂度: $O(n(n+m)C)$. 注: pseudo polynomial!

原因: 若用二进制表达 C, 则实际运行次数为其位数的指数级别.

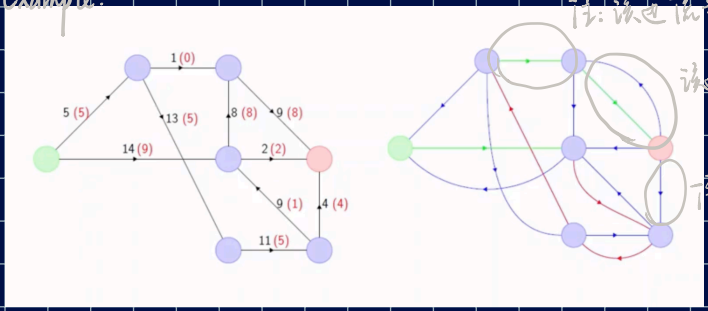
- Suppose we have a network with
  - n vertices
  - m edges
  - maximum edge capacity C.
- We know that REACHABILITY can be solved in $\text{TIME}(O(n+m))$ (optimized union-find) where n is the number of nodes and m the number of edges.
- The value of any flow $\leq nC$,
- Hence the operating time is at $O(n(n+m)C)$.

原因: 在 worst case 中需执行 nC 次操作才可得到 maximum flow. 此时 optimal flow 恰为 nC.

第一次操作时找一次 path, 耗时 $O(n+m)$

$\Rightarrow$ total time complexity: $O((n+m)nC)$.

Example:



注: 该边流量为0, 故不存在反向弧.

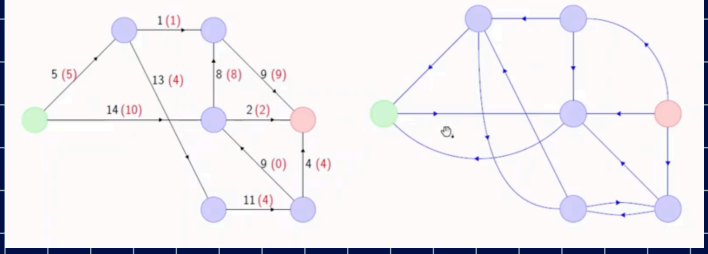
该边流量<容量, 此时正、反向弧均存在.

注: 该边流量=容量, 故不存在正向弧.

和: 左图为当前给定的 flow network. 右边即为基于当前的 network flow 所得到的 augmented graph.

红和绿边即为在 augmented graph 中找到的, 从顶点到汇点的 path.

其中, 绿边是 positive path, 而红边为 negative path.



然后执行 Ford-Fulkerson: 对找到的 path 中每一条边:

- 对 positive path: 流量加一单位.
- 对 negative path: 给它在现实流网络图中对应的边减一单位流量.

$\Rightarrow$ 得到的结果如上图所示. //

对 Min-cut, Max-flow 正确性的证明:

Lemma (Min-cut, Max-flow)

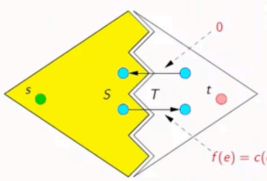
Let $N = (V, E, s, t, c)$ be a flow network and f a flow in N. Then there is a path in N_f from s to t if and only if f is not optimal.

"$\Rightarrow$" 由 Ford-Fulkerson 等价知: 当 N_f 中无增广路径时, f 确为 optimal 的.

"$\Leftarrow$": 设 N_f 中无增广路径, 要证明: f 为 optimal.

由于 N_f 中无增广路径, 故对流网络图作切割:

The only-if direction is trivial. Suppose, conversely, there is no path from s to t in N_f. Let S be the set of nodes V reachable from s in N_f, and $T = V \setminus S$.



知: 由于对图中任一个连接切割 S 和 T 的边, 其在 N_f 中不存在从 S → T 的正向弧: (assumption)

- case 1: 在原图中有一条从 S → T 的弧, 但 capacity = flow.
- case 2: 在原图中有一条从 T → S 的弧, 但 flow = 0.

for either case, there's no spare capacity from S to T.

$\Rightarrow$ 得证. //

2D- Matching 问题 (即: bipartite graph 的加权匹配问题)

example: a list of boys and girls, each one has their "preferred companions". find a 1-1 matching between boys & girls, with maximal pair num.

引入 Definition (bipartite graph):

Bipartite Graph: a triple $G = \langle U, V, E \rangle$: U, V 为对顶点集的一个分划; 并且显然 $E \subseteq U \times V$.

建模匹配问题：

- Let $G = (V, W, E)$ be a bipartite graph. A matching is a subset $E' \subseteq E$ such that E such that for all $v \in V$, there is at most one $w \in W$ with $(v, w) \in E'$, and, for all $w \in W$, there is at most one $v \in V$ with $(v, w) \in E'$. 最多1个 ⇒ 1~1对应.
- The matching is perfect if every node in V and W is incident to some $e \in E'$.

关键知：对于二部图的匹配，有以下的（形式化）的问题：

MAXIMAL-MATCHING
Given: A bipartite Graph G
Return: A matching of maximal cardinality.
(maybe not perfect)

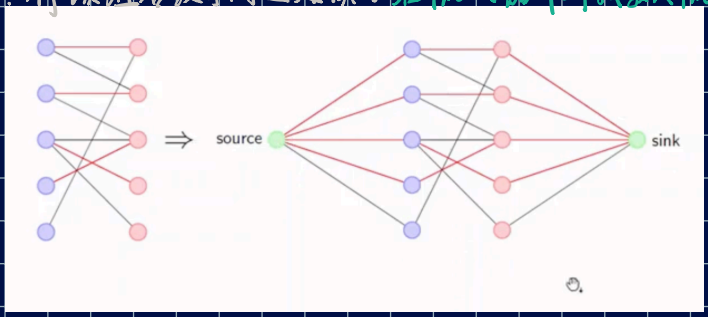
1. find the maximal matching of a given graph.

MATCHING
Given: A bipartite Graph G
Return: Yes if G has a perfect matching, and No otherwise.

2. Determine if given bipartite graph has a perfect matching.

解法：
1. Naive Match：尝试每一种可能的 matching。可知：时间复杂度为指数级别。

2. 将该组合数学问题建模为在流网络中寻找最大流的问题：



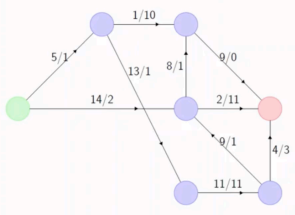
给定二部图：
1. add a source and a sink node.
connect source (sink) with every node in U (V).
2. set the capacity of each edge to be 1.
3. use ford-fulkerson algorithm to compute the maximal flow.

since: { 1. in (2) every edge's capacity is 1
2. the construction of flow network prevent multiple-to-one matchings.

⇒ the value of maximal flow is the cardinality of the maximal matching.

Minimal-Cost 问题：关注"如何得到成本最低的最优网络流"。关注流量和成本（distance）。
引入新定义：flow network with costs.
(distance)

- A flow network with costs is a sextuple $N = (V, E, s, t, c, \gamma)$ where (V, E, s, t, c) is a flow network and $\gamma : E \to \mathbb{N}$.



注：现在 each edge has its capacity and cost for connecting.
此处的 cost 是单位流量的连接成本！

- If f is a flow, the total cost of f is $\sum_{e \in E} f(e) \cdot \gamma(e)$.

2) 增广图中的反向弧的 cost 为其在求流网络图中对应边的 cost 的取反！

- if (u, v) is an edge of N_f and $(u, v) \in E$, the distance $d(u, v)$ is $\gamma(e)$;
- if (u, v) is an edge of N_f and $(v, u) \in E$, the distance $d(u, v)$ is $-\gamma(e)$.

数学解决问题：

MIN COST MAX FLOW 找 max flow 中成本最小的.
Given: A flow network with cost function, N
Return: An optimal flow f for N having the minimum cost among all optimal flows.

解决方法：利用 Ford-Fulkerson 等流的修改版本：Busacker-Gowen Algorithm.

1. 理论基础：

Lemma

Let N be a flow network with cost function, and suppose f is a flow of value v through N, such that f has minimal cost among all flows of value v. Suppose π is a path in N_f from s to t of minimal length, and let f' be obtained from f by augmenting along π (in the usual way). Then f' has minimal cost among all flows of value $v+1$.

2. 算法描述：

- The Busacker-Gowen algorithm for computing minimum cost optimal flows:

```
begin flowMaxCost(V,E,s,t,c,γ)
    set f(e) = 0 for all e ∈ E
    while reachable(N_f,s,t)
        let e_1,...,e_m be the edges on a shortest path from s to t
        for 1 ≤ i ≤ m
            if e_i ∈ E, increment f(e_i)
            else decrement f(e_i^{-1})
    return f
```

the only difference

注：
1) 等次不改变的 shortest 性质
2) 第次循环，value +1

Idea: 1. start from "minimal cost flow with value 0"
0. then start augmenting the shortest augmented path in the augmented graph of this flow
3. get a new flow with minimal cost and value 0+1 = 1
4. go back to (2) and keep exploiting until no path can be found in augmented graph.

⇒ 此时得到的网络流 f 即为最大流中 cost 最小的.

注：由于增广图中可能有负权边 (回忆：最短路的 distance ?)，故应用支持负权边的 Bellman-Ford 算法找增广图中的最短路径.

§VI. Matching with Preferences.

问题：在每个个体都有 preferences 的 bipartite graph matching 问题基础上：

Suppose matched individual has preferences expressed as rankings.

将问题形式化：引入 "stability" 的定义。

目标：在多项式时间内计算 stable matching。

1. Stable Matching Problem:

- We are given:
  - a set of *n* boys and *n* girls;
  - a strict ranking, for each boy, of all the girls
  - a strict ranking, for each girl, of all the boys
- We want to compute:
  - a 1–1 pairing of boys with girls in which, for every boy *a* and girl *b*, either *a* prefers his partner to *b* or *b* prefers her partner to *a*. (Such a pairing is called a stable matching.)

注：a stable matching always exists.

2. 计算 stable matching：Gale - Shapley 算法。

The Gale-Shapley algorithm generates a matching as follows

```
begin Gale-Shapley(Boys' rankings, Girls' rankings)
  until all boys are engaged do
    for each boy a with no fiancée do        propose to the girl that he most prefers
      a proposes to the girl he most prefers among
      those that a has not yet proposed to
    for each girl b with new proposals do    compare its fiance (if exists) with the new proposal holder and
      let a be b's most preferred new suitor  select the more optimal one between the two
      if b has no fiancé
        b gets engaged to a
      if b prefers a to her existing fiancé
        b cancels her existing engagement
        b gets engaged to a
  All the engaged couples get married
the end
```

Initial state：假设所有人均未匹配。

1. every boy propose to the girl with highest preference.
2. every girl select the most optimal boy from all her proposals.
3. if any boys are rejected, they delete the girl who rejects them from their preference list, and repeat the same procedure as (1):

until none the boys are rejected.

Now the matching is stable.

Theorem

The Gale-Shapley algorithm terminates with a stable matching.

Proof:

1. 可知算法最终必会 terminate，因为形成的是一个 1-1 配对。

   并且知：算法结束时，每个 girl 均会收到 proposal ~ 一定求进了形成某一个 matching（必然从 proposals 中选择了 fiancee）

   ⇒ 故形成了匹配。

2. 下面证明：该匹配是 stable 的：

   不妨假设 $\langle a, b' \rangle$ 和 $\langle a', b \rangle$ 为所生成的两个匹配，下面说明它们均 stable：

   若实际上 b 更垂青于 a，而仍然生成了这样一组匹配，则说明男孩 a 从未 propose to b，否则必形成了 $\langle a, b \rangle$.

   由于 a 向所有对他的言语多于和 b' 一样好的女孩 propose，故说明 a 不认为 b 比 b' 更好

   ⇒ a prefers b' over b. 满足 "matching stable" 的条件。//

故知：形成了 stable matching。//

3. Gale-Shapley 算法的时间复杂度：$O(n^2)$，其中 n 为 # of boys (girls)。

4. Gale-Shapley 算法的一个性质：

Theorem

The stable matching, M, produced by the Gale-Shapley algorithm is optimal for boys: if boy a is married to girl b in M, but prefers girl b′, then there is no stable matching M′ in which a is married to b′.

(there's no better matching for boy 'a').

换言之：生成的关于 'a' 的匹配总是对 'a' 在给定条件下的最优解。

Proof:

Definition 1. *A man m and a woman w are **valid partners** means there exists a stable matching in which they are paired with each other.*

Definition 2. *For every man m, m's **best valid partner** (denoted $best(m)$) is the highest-ranked valid partner of m, with respect to m's preference list.*

Definition 3. *A matching S is **man-optimal** means that each man m is paired with $best(m)$ in S.*

We have already seen that the Gale-Shapley algorithm (GS) always returns a matching that is stable. We will need that to prove the following.

Claim 1. *For any fixed rule dictating the order of proposals, the matching S^* returned by GS is man-optimal.*

Proof. First note that since S^* is stable, everyone is paired with one of their valid partners. Suppose by way of contradiction that there exists a man who is not paired with his best valid partner. Then he must be paired with a valid partner who comes after his best valid partner in his preference list. In other words, if S^* is not man-optimal, then at least one man was rejected by his best valid partner during the execution of GS.

Consider the *first* time it happens that some man m is rejected by his best valid partner, $w = best(m)$: w rejects m to be (or to continue to be) with someone else m' whom she prefers to m. Let us call this episode Event X.

Since w and m are valid partners, there exists a stable matching S' in which w is paired with m. In S', m' is paired with someone else, say $w' \neq w$. w' is a valid partner of m' since S' is stable.

Now consider what the execution of GS tells us about m''s preference between w and w'. **Event X was the first time in the execution of GS that any man was rejected by his best valid partner. In particular, at the time that Event X occurred, both the following are true:**

- **m' has not been rejected by his $best(m')$ and therefore has not been rejected by any of his valid partners, in particular w';**
- **m' is paired with w, i.e. m' was rejected by every woman before w in his preference list.**

Therefore, the execution of GS tells us that w' must be after w in m''s preference list, i.e. m' prefers w to w'.

However, this contradicts the stability of S': $(m, w), (m', w') \in S'$, but both w and m' prefer each other to their respective partners in S'. Therefore, our initial assumption that some man is rejected by his best valid partner during the execution of GS is false, i.e. S^* is man-optimal.

□

§VII. Machine Stop.

问题: { 1. Decidability
2. Time & Space Complexity

⇒ 建立通用的抽象计算模型 (Turing Machine) 并说明存在非确定性问题.

形式化的计算模型: Turing Machine.

* Definitions:

1. Turing Machine:

- Formally, a Turing Machine is a quintuple

$$M = \langle K, \Sigma, Q, q_0, T \rangle,$$

Turing Machine定义为一个可被使用素数编码法所编码的元组
可用于模拟任何可能的计算

where

- $K \geq 2$ (number of tapes) input, output, intermediate, ...
- Σ is a non-empty, finite set (alphabet)
- Q is a non-empty, finite set (set of states)
- $q_0 \in Q$ (initial state)
- T is a set of transitions (for K, Σ and Q)—see below.

2. Symbol:

任何图灵机中都包括起始符号和终止符号

A symbol is an element of $\Sigma \cup \{\sqcup, \triangleright\}$

- We pronounce $\sqcup$ as blank and $\triangleright$ as start .
- We denote the set of finite strings over Σ by Σ^*.

3. Transition:

定义转换: 条件+ 结果.
条件: 位于给定状态下, 满足特定条件的情况下可以发生;
结果: 从当前状态转换到的下一状态; 在下一状态下对纸带进行了什么修改.

- A transition (for K, Σ and Q) is a quintuple

$$\langle p, \bar{s}, q, \bar{t}, \bar{d} \rangle$$

where

- $p \in Q$ and $q \in Q$ states
- $\bar{s}$ and $\bar{t}$ are K-tuples of symbols condition + target word of writing
- $\bar{d}$ is a K-tuple whose elements from $\{\texttt{left}, \texttt{right}, \texttt{stay}\}$ operation

- Informal meaning:
  If you are in state p, and the symbols written on the squares currently being scanned on the K tapes are given by $\bar{s}$, then set the new state to be q, write the symbols of $\bar{t}$ on the K tapes and move the heads as directed by $\bar{d}$.

p: 当前所处状态.
$\bar{s}$: 在全部 K 条纸带上的当前读出符号元组
q: 将转移到的新状态.
$\bar{t}$: 将在全部 K 条纸带上写入的符号元组
$\bar{d}$: 对于每一条纸带, 相对应磁头在写入后的移动方向.

注: 1. tape never move past 'start ▷'.
2. tape 1: input tape; tape K: output tape;
(read only) (write only)
3. 'start ▷' will never be overwritten
nothing is overwritten with '▷' or '␣'. ?

4. Configuration (格局):

图灵机的"格局"实际上就是在某个特定状态下对它执行状态的快照.

对它的每一条纸带, 记录纸带所存储的数据和磁头当前所在的位置, i.e. 程序所执行到的状态.

A configuration of M: a K-tuple of strings $\sigma_1, \ldots, \sigma_K$, where σ_k $(1 \leq k \leq K)$ is of the form

$$\triangleright, s_{k,1}, \ldots, s_{k,i-1}, q, s_{k,i}, \ldots, s_{k,n(k)}.$$

This states that the kth tape of M reads $\triangleright, s_{k,1}, \ldots, s_{k,n(k)}$, the head is over square i, and the current state is q (same for all K strings).

5. Run (运行):

图灵机的某次运行 ⇒ 记录为一系列格局 (running snapshots)

A run of machine M on input x is a sequence of configurations (finite or infinite) in which successive configurations conform to some transition in T, and such that, if some transition is possible in a configuration of the run, that configuration cannot be final.

6. Finite Run:

a run is finite ⇔ it's terminating.

7. Deterministic:

记 Turing Machine 是 deterministic 的，若对任何的当前所处状态 p 和在全部 k 条纸带上的当前读出符号元组 $\vec{s}$，有唯一对应的转换：$\langle p, \vec{s}, q, \vec{t}, \vec{d} \rangle$.

并且约定记号如下：

- $M \downarrow x$ means that the run of M on input x is terminating
- $M \uparrow x$ means that the run of M on input x is non-terminating

"the run": 由考虑的范围只能是确定性图灵机而决定

给定确定性图灵机的一个输入 x, 必然只有一种run

8. "Computes" 和 "Computable"

记 M 为一个定义于 alphabet: Σ 上的 Deterministic Turing Machine.

若 M 关于输入 x 的 run 为 terminating 的，i.e. $M \downarrow x$：则称 M 的 output tape 上必含有一个长度有限的 string: $y \in \Sigma^*$.

故由此类推，对于 M 的任何 input $x \in \Sigma^*$，可定义如下的 partial function:

$$f_M(x) = \begin{cases} y & \text{if } M \downarrow x \\ \text{undefined} & \text{otherwise} \end{cases}$$

并且称：M computes f_M (本质上 Turing Machine M 可被抽象为 f_M).

同时，称某个 partial function f_M 为 computable 的，若它 is computed by some deterministic Turing Machine.

* 利用 prime-power coding 将 Turing Machine 编码为某个数字：

- Q. How do you encode a sequence of numbers $m_1, m_2, m_3, \ldots, m_k$ as a single number?
- A. For example, as as $2^{m_1} \cdot 3^{m_2} \cdot 5^{m_3} \cdot \quad \cdots \quad \cdot p_k^{m_k}$ (prime-power coding).
- Suppose M is a Turing machine with code m. Then a possible input to a(nother) Turing machine is $m; x$.

可将图灵机使用素数编码法编码为某个数字

* Theorem:

给定一个字母表 Σ. 存在某个 Turing Machine U, 满足：

for $\forall$ turing machine M over Σ, with code m :

for $\forall$ string $x, y \in \Sigma^*$:

1. U has a terminating run on input $(m; x)$ leaving y on output tape
   ⇔ M has a terminating run on input x, leaving y on output tape.

2. U has a non-terminating run on input $(m; x)$
   ⇔ M has a non-terminating run on input x.

* 下面引入更多的 Definitions:

1. Acceptance & Recognition :

记 M 为一个定义于字母表 Σ 上的 Turing Machine. 令 string $x \in \Sigma^*$.

称：M accepts x, 若 M has a halting run on input x with the first head over leftmost square.

(why important?)

并且记由所有被 M 所 accept 的 string 构成的 set 为：the language recognized by M.

2. Recursively Enumerable: (r.e.)

称 a language is recursively enumerable 若 ∃ 一个 Turing machine M, that recognize this language.

3. Co-recursively Enumerable: (co-r.e.)

若 $\Sigma^* \setminus L$ is recursively enumerable, 则称 language L 为 co-recursive enumerable.

4. Recursive:

称 language L 为 recursive 的, 若它同时 r.e 和 co-r.e.

(Recursive: 关于 language 的性质: L 本身和它关于给定字母表生成的补: $\Sigma^* \setminus L$ 均被某个 Turing machine 所识别.)

* Theorem:

if a language is recognized by some Turing Machine M ⇒ it's recognized by some deterministic Turing Machine M'.

* Theorem:

a language is recursive ⇔ ∃ a deterministic Turing machine M that recognize it.

Pr. (personal):

L is recursive ⇔ L is r.e & $\Sigma^* \setminus L$ is r.e ⇔ ∃ TM: M_2, s.t. M_2 recognizes $\Sigma^* \setminus L$ ⇒ ∃ det. M_2', s.t. M_2' recognizes $\Sigma^* \setminus L$

⇕

∃ TM: M_1, s.t. M_1 recognizes L. ⇒ ∃ det. M_1', s.t. M_1' recognizes L.

⇒ easy to notice: det. TM: M_1' computes: $f_{M_1'}(x) = \begin{cases} \text{True} & x \in L \\ \text{False} & x \in \Sigma^* \setminus L. \end{cases}$

then just let M_2' be $\neg M_1'$, computes: $f_{M_2'}(x) = \begin{cases} \text{True} & x \in \Sigma^* \setminus L \\ \text{False} & x \in L. \end{cases}$ //

接着下面的结论:

Sometimes, we speak of (decision) problems and decidability:

- language ⟺ (decision) problem
- recursive ⟺ decidable

recursive问题实际上是一个决策问题
程序需要判断 (决定) 给定的句子 X 是否在语言中
本质上也就是一个输出 "yes or no" 的问题

若将 language 视为一类
需要给出答复的 decision problem:

则实际上, 所谓的 recursive 就是指:
这类决策问题是否为 decidable 的.

下面给出 decision problem 的形式化表述:

PROBLEM NAME
Given: a string x (coding some object we are interested in);
Return: Yes if x has some property $\mathcal{P}$, and No otherwise.

(即: answer yes, no or unknown)

The Halting Problem:

HALTING
Given: a pair of strings m, x;
Return: Yes if m is the code of a deterministic Turing Machine, M, and x a string in the alphabet of M, such that $M \downarrow x$;
No otherwise.

即: 是否所有的 decision problem 类 (languages) 都是 decidable (recursive) 的?

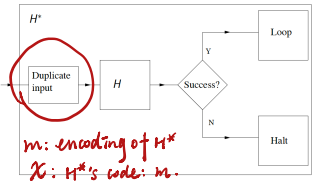
(does it exists any undecidable problems?)

结论: Halting Problem is not decidable.

证明: 构造一个以某个 Turing Machine 的 code 和 input 为 input 的, 又不同 Turing Machine 作 wrapper, 并得到 contrary.

Proof.
Suppose H is a deterministic TM such that, for every deterministic TM M with code m, and every string x in the alphabet of M, H outputs Yes on input $(m; x)$ if $M \downarrow x$, and No otherwise. Let be H^* as below, with code h^*.

此处的 H*是一个更大的图灵机，used as a wrapper; its input is h*, the encoding of itself



m: encoding of H*
X: H*'s code: m.

What happens if H^* is given input h^*? The embedded H receives input $h^*; h^*$. Hence:

$$H^* \downarrow h^* \Rightarrow H^* \uparrow h^* \quad H^* \uparrow h^* \Rightarrow H^* \downarrow h^*$$

all contrary ⇓ such H doesn't exist!

§VIII Time & Space Complexity

目标：1. 对算法运行的时间&空间复杂度来进行定义和分级.
2. 如何计算出 deterministic / non-deterministic Turing Machine 运行的时空复杂度.

Definition: runs in time (time consumption) 和 use space (space consumption)

Let M be a Turing machine with alphabet Σ, and let $g : \mathbb{N} \to \mathbb{N}$. We say M runs in time g if, for all but finitely many strings $x \in \Sigma^*$, any run of M on input x halts within at most $g(|x|)$ steps. Similarly, M runs in space g if M always terminates and, for all but finitely many strings $x \in \Sigma^*$, any run of M on input x uses at most $g(|x|)$ squares on any of its work-tapes.

⇒ 形式化地定义 "runs in time" 和 "runs in space":

Turing Machine M runs in { time g ⇔ 对 $\forall x \in \Sigma^*$, 执行步数最多为 $g(|x|)$ 步.
space g ⇔ 对 $\forall x \in \Sigma^*$, M 上任一条 tape 占用了最多 $g(|x|)$ 个格.

Definition: 对 Language 的归类："问题的 Deterministic 时空复杂度类"

Let L be a language over some alphabet, and let $g : \mathbb{N} \to \mathbb{N}$ be a function. We say that L *is in* $\textsc{Time}(g)$ if there exists a deterministic Turing machine M recognizing L such that M runs in time g.

Let L be a language over some alphabet, and let $g : \mathbb{N} \to \mathbb{N}$ be a function. We say that L *is in* $\textsc{Space}(g)$ if there exists a deterministic Turing machine M recognizing L such that M runs in space g.

given Language L:

L in Time (g) ⇔ ∃ turing machine M recognize L, runs in time g.

L in Space (g) ⇔ ∃ turing machine M recognize L, runs in space g.

Theorem (Speed-Up)

Linear 'speed-up' theorems:

Theorem ($c \in (0,1)$, linear factor.)
Let $f : \mathbb{N} \to \mathbb{N}$ be a function, and $c > 0$. Then $\textsc{Time}(f(n)) \subseteq \textsc{Time}(c \cdot f(n) + n)$.

Theorem
Let $f : \mathbb{N} \to \mathbb{N}$ be a function, and $c > 0$, $\textsc{Space}(f(n)) \subseteq \textsc{Space}(c \cdot f(n))$.

Language recognizable by a (deterministic time/space) TM in $f(n)$ is contained by language recognizable by a (deterministic time/space) TM in $c \cdot f(n) + n$ (c·f(n) space) (quicker time or smaller space)

idea: build up a turing machine (TM) with a larger alphabet, larger set of states
⇒ compile several steps in original TM into a single step in the new TM.

规定：一般简化记录 languages 的 time / space complexity.
并且有以下的常用记法：

- Let G be a set of functions from $\mathbb{N}$ to $\mathbb{N}$:

consider: time and space complexity bounded by a set of functions 得到的是 "时空复杂度类"

$$\textsc{Time}(G) = \bigcup_{g \in G} \textsc{Time}(g)$$
$$\textsc{Space}(G) = \bigcup_{g \in G} \textsc{Space}(g).$$

- Here are some handy function-classes:

P: Polynomial
E_i: Exponent of 2 for i times..

$$P = E_0 = \{n^c \mid c > 0\}$$
$$E = E_1 = \{2^{n^c} \mid c > 0\}$$
$$E_2 = \{2^{2^{n^c}} \mid c > 0\}$$
$$E_k = \{2^{2^{\cdots^{2^{n^c}}}} \text{ (k times)} \mid c > 0\}$$

注：Time (g) ~ languages bounded by a specific function g.

Time (G) ~ languages bounded by a function class G.

关于 Space (G) 和 Space (g)：同理.

P: polynomial
E_i: exponent of 2 for i times.

同时还有：

They are ALL recognizable by some deterministic turing machine which running time bounded by xxxxx function (class)

$$\textsc{PTime} = \textsc{Time}(P)$$
$$\textsc{ExpTime} = \textsc{Time}(E)$$
$$k\text{-}\textsc{ExpTime} = \textsc{Time}(E_k)$$

Note: for time complexity every complexity class is more complex than Time(n) since if time is shorter it won't be able to even read in the input (given input's length n)

注：1. languages ⇔ set of (decision) problems
2. each set of problems : defined by TIME / SPACE bound for the deterministic TM to recognize them.

相对应地，关于 space complexity:

$$
\begin{aligned}
\text{LogSpace} &= \text{Space}(\log n)\\
\text{PSpace} &= \text{Space}(P)\\
\text{ExpSpace} &= \text{Space}(E)\\
k\text{-ExpSpace} &= \text{Space}(E_k)
\end{aligned}
$$

=> complexity class: sets of languages !

关于一些知名问题，其时间复杂度类：

1. Problems we know are in PTime
   - st-CON. — reachability of directed graph (use DFS)
   - undirected st-CON — reachability of undirected graph (use DFS)
   - CYCLICITY — determine if graph is cyclic
   - 2D-MATCHING — determine whether exists a perfect matching in a bipartite graph

2. ALSO IN PTime!
   线性规划可行性问题

   LINEAR PROGRAMMING FEASIBILITY (LPF)

   Given: A system $\mathcal{E}$ of linear inequalities $A\bar{x} \leq \bar{b}$.
   Return: Yes if $\mathcal{E}$ has a solution over $\mathbb{R}^+$, No otherwise.

   will not discuss proof...

3. **PRIMES**
   Given: An integer n
   Return: Yes if n is prime, No otherwise.

   ALSO IN PTime!
   素数检测问题

   WARNING:
   'numerical input is given as BIT STRINGS
   input n => length of its bit string form is log(n)

   M. Agrawal, N. Kayal and N. Saxena, 2004: PRIMES is in PTime

   will not discuss proof...

注：和 Ford-Fulkerson 与 Bulsacker-Gowen 算法一样：
只有我们关注（max capacity）输入的 integer 的值
即非其表示的位数时，它们的时间复杂度才为 polynomial.

以及两个重要的空间复杂度类结论：

1. any language generated by context-sensitive grammar is in Space(n) ⊆ PSpace (polynomial space).

2. Undirected st-CON (无向图的 reachability 问题) is in LogSpace.

下面关注 non-deterministic 的情形：

Definition：对 Language 的归类："问题的 Non-Deterministic 时空复杂度类".

Definition (NTime: NON-DETERMINISTIC Time; NSpace: NON-DETERMINISTIC Space)

Let L be a language over some alphabet, and let $g : \mathbb{N} \to \mathbb{N}$ be a function. We say that L *is in* $\text{NTime}(g)$ if there exists a Turing machine M recognizing L such that M runs in time g.

这里提到的Turing Machine其实是Non-deterministic Turing Machine
As Turing machines are non-deterministic in general

Let L be a language over some alphabet, and let $g : \mathbb{N} \to \mathbb{N}$ be a function. We say that L *is in* $\text{NSpace}(g)$ if there exists a Turing machine M recognizing L such that M runs in space g.

$$\text{NTime}(G) = \bigcup_{g \in G} \text{NTime}(g)$$

$$\text{NSpace}(G) = \bigcup_{g \in G} \text{NSpace}(g).$$

相应地有：

non-deterministic exponential time
non-deterministic to the...
...exponential k time

$$
\begin{aligned}
\text{NPTime} &= \text{NTime}(P)\\
\text{NExpTime} &= \text{NTime}(E)\\
k\text{-NExpTime} &= \text{NTime}(E_k)
\end{aligned}
$$

similar to definitions in deterministic turing machine's case:
the language is accepted by some NON-deterministic
turing machine with a time/space bound of polynomial, exponential funs, etc.

$$
\begin{aligned}
\text{NLogSpace} &= \text{NSpace}(\log n)\\
\text{NPSpace} &= \text{NSpace}(P)\\
\text{NExpSpace} &= \text{NSpace}(E)\\
k\text{-NExpSpace} &= \text{NSpace}(E_k).
\end{aligned}
$$

另一些和 NTime, NSpace 相关的结论：

1. st-CON (有向图中的 reachability 问题) is in NLogSpace：
   证明：给定一个 non-deterministic 的，in NLogSpace 的"算法"：

```
begin reachND(G, u, v)
    n := number of vertices in G
    w := u
    c := 0
    while w ≠ v and c < n
        pick w' such that w = w' or there is an edge from w to w'   (guess the vertex and check)
        w := w'
        increment c;
    if c < n
        return Y                                                   (must got to the destination)
    return N                                                       (if the guessing fails then return)
```

注：若 tree 的 size 为 n，则其 height 为 log n，作为 worse case 下
"找到"（若能找到的话）的 path 的长度.

2. TSP 问题（旅行商问题）：

TSP feasibility
Given: a symmetric matrix A with non-negative integer entries;
Return: Y if there is a tour of length $\leq k$ in A;
N otherwise.

find the shortest path under the constraint of "must-visit" nodes

Is in NPTime: 给定一个 tour，可以在多项式时间内验证。

Theorem
TSP feasibility is in NPTIME.

Proof.
Here is an 'algorithm':

```
begin TSPfND(A, k)
    Guess a tour π in A.
    Let ℓ := the length of π
    if ℓ ≤ k
        return Y
    return N
```

What is "non-deterministic polynomial time"

i.e. for this example, how can we infer the running time is bounded by some polynomial given the fact that we are randomly guessing?

=> if the problem in deed has a solution and you happens to guess that solution, then it will halt in polynomial time!
=> if the problem has no solution then the turing machine will never stops
==> fit the characteristic of non-deterministic turing machine!

3. k-COLORABILITY (图的 k-染色问题) Is in NPTime:

k-COLOURABILITY
Given: A graph G.
Return: Yes if G is k-colourable, and No otherwise.

How to prove? Similar to TSP: just guess!

1. if problem has solution and guessed the solution => can be validated in polynomial time
2. if problem has no solution => runs never halts

证：
1> k-coloring: a function $f: V \to [0, 1, \cdots, n-1]$, s.t. for $\forall (u,v) \in E$ (edge), $f(u) \neq f(v)$.
2> graph $G = \langle V, E \rangle$ is k-colorable 当 $\exists$ a k-coloring for it.

下面讨论不同时空复杂度类之间的 inclusion (包含关系)：

1. 首先有：

- The following inclusions are obvious:

$$\text{TIME}(G) \subseteq \text{NTIME}(G) \qquad \text{SPACE}(G) \subseteq \text{NSPACE}(G)$$
$$\text{TIME}(G) \subseteq \text{SPACE}(G) \qquad \text{NTIME}(G) \subseteq \text{NSPACE}(G)$$

- Also, if $G \subseteq H$, then $\text{TIME}(G) \subseteq \text{TIME}(H)$, and similarly for NTIME, SPACE and NSPACE.

显然.

☆重要性质：
$\begin{cases} \text{Time}(g) \subseteq \text{Space}(g) \\ \text{NTime}(g) \subseteq \text{NSpace}(g) \end{cases}$

take some $g \in G$ as space bound:
it means the maximum tape square usage is bounded by g
⇒ to fill/scan these squares, # of steps "$\geq g$".
⇒ (languages with space bound Space(g)'s time bound $\geq$ Time(g).)

2. 其次定义 Configuration Graph:
考虑 Turing Machine $M = \langle k, \Sigma, Q, q_0, T \rangle$:
1> 设其 running space bounded by function 's'.
2> 设：对长为 n 的任何输入，该 Turing Machine 最多有

$$|Q| \cdot |\Sigma|^{k \cdot s(n)} \cdot (s(n)+1)^k \in 2^{O(s(n))}$$

(|Q|: 当前state; $|\Sigma|^{k \cdot s(n)}$: 所有k条tape上的charlists; $(s(n)+1)^k$: tape head 当前位置(状态+指下一位))

个 configuration.
3> 构建 Configuration Graph: $G = \langle V, E \rangle$ 将每个 configuration 视为一个 node $\in V$,
若从 configuration α 可转到 β, 则 $(\alpha, \beta) \in E$. (假设无环)
4> 由此：给定输入 x，就可确定关于 x 的 start configuration，记为 C_0，和 success configuration，记为 C_*.
5> 故对于给定输入 x，若 G 中有在从 C_0 到 C_* 的 path，则说明该 TM accepts x.

等同称 G: configuration graph for M with input x.

和 Configuration Graph 相关的一个 Thm:
☆ Theorem: $\text{NSpace} \subseteq \text{Time}(2^{O(g)})$.:
$\text{NSpace} \subseteq \text{Time}(2^{O(g)})$.

证明：Let M in NSpace(g). 目标：说明 M in Time($2^{O(g)}$).
给定任何 input x, $|x| = n$, 关注 M, x 为输入的 Configuration Graph:

1) 构建 G 的空间复杂度不超过 n，因对其的每一位 pare 对应一个 state.

2) 即由于 $|G| < 2^{O(g(n))}$：在 worst case 下，在 G 中搜索从 starting position with input x 到 success position 的 path，至多耗时 $C \cdot |G| \leq C \cdot 2^{O(g(n))}$，即 M 关于输入 x 的 running time $< 2^{O(g)}$!

$\Rightarrow$ M in Time $(2^{O(g)})$. //

故有: $NSpace \subseteq Time(2^{O(g)})$.

由该 Thm 立刻知:

$NLogSpace \subseteq PTime$, $NPSpace \subseteq ExpTime$,
$NExpSpace \subseteq 2\text{-}ExpTime$, *etc.*

并且有以下的 Thm:

☆ Theorem:

$NTime(g) \subseteq Space(g)$.

证明: 为证任一个 $L \subseteq NTime(g)$，下面证明到等价的确的算法 is in Space (g).

设 L is recognized by a non-deterministic TM: M, 知其 runs in time g.

故知: M 对输入 n，作出了最多 $g(n)$ 个 random choice，并设所有可能的 random choice 组成一个 list，记该 list 大小为 k.

由此 M 的一次运行可记为: $k_1, \cdots, k_{g(n)}$：每一步均在 k 个选项中随机选一个，由此"形成了一个 run".

现在关注 run 的生成:

构造一个含有一条记录了 M 运行时 k 个 random choice 依序的 deterministic TM: M^*.

则通过 cycle through all possible sequences $k_1 \cdots k_{g(n)}$，我可在利用最多 $g(n)$ 空间的情况下生成 M 的每种 run (总含有被 M 所 recognized 的)

$\Rightarrow NTime(g) \subseteq Space(g)$. //

故结合一个结论和两个 Thm, 有:

$$LogSpace \subseteq NLogSpace \subseteq PTime \subseteq NPTime \subseteq PSpace \subseteq NPSpace \subseteq ExpTime \subseteq NExpTime \subseteq ExpSpace \subseteq NExpSpace \subseteq 2\text{-}ExpSpace \subseteq \cdots$$

—: 结论 —: Thm 1: $NSpace(g) \subseteq Time(2^{O(g)})$ —: Thm 2: $NTime(g) \subseteq Space(g)$.

最后引入两个 "$\subsetneq$" 关系:

Thm 3.

for all time-constructable, increasing functions $f(n)$ on n: $Time(f(n)) \subsetneq Time(f(2n+1)^3)$.

证明: 首先说明: $HALTING_f \notin Time(f(\lfloor \frac{n}{2} \rfloor))$.

下面引入 $HALTING_f$:

Definition
Let $f : \mathbb{N} \to \mathbb{N}$ be a function. The f-bounded Halting problem is the following problem:

$HALTING_f$
Given: the code, m, of a deterministic Turing Machine, M, and a string, x, in the alphabet of M;
Return: Yes if M terminates on input x in time at most $f(|x|)$, and No otherwise.

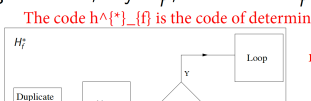
Note: problem HALTING_f is stronger than HALTING

want to know:
1. whether M halts at input x
2. if its running time bounded by f.

并证明 $HALTING_f \notin Time(f(\lfloor \frac{n}{2} \rfloor))$:

Proof. 使用反证法
Suppose $HALTING_f$ is recognized by a Turing machine H_f, guaranteed to terminate in time $f(\lfloor n/2 \rfloor)$.
Consider the Turing machine, say H_f^*, with code h_f^*:

The code h^{*}_{f} is the code of deterministic turing machine H_f



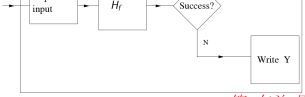
not terminate at all

使用证明HALTING问题的相似思路: 构造一个以假设的图灵机自

由于它 not recognize $HALTING_f$:

身的编码作为输入的另一个确定性图灵机

然后推出矛盾



terminating instantly

$H_f^* \downarrow h_f^* \Rightarrow H_f^* \uparrow h_f^*$
$H_f^* \uparrow h_f^* \Rightarrow H_f^* \downarrow h_f^*$

问题: 这个证明和f的形式有关吗? 如果换成诸如f(n^2)的形式行不行?

答: 有关, 因为在 'Duplicate Input' 处, 将作为输入的编码长度增倍, 因此需要确保接收长度增倍后的输入的图灵机H_f运行时间为这个形式, 否则就无法推出矛盾

故这样的 Turing Machine 不存在.

$\Rightarrow$ HALTING$_f \notin$ Time($f(\lfloor n/2 \rfloor)$).

关于 time-constructable 的定义: 不等.

用结论:

If f is time-constructible, increasing and reasonably fast-growing, so is $f'(n) = f(2n+1)$. Moreover, $f(n) \leq f'(\lfloor n/2 \rfloor)$. $U_{f'}$ decides HALTING$_{f'}$, and runs in time $f'(n)^3 = f(2n+1)^3$. On the other hand, HALTING$_{f'}$, is not computable in time $f'(\lfloor n/2 \rfloor) \geq f(n)$. Hence:

$\Rightarrow$ Theorem

For all time-constructible, increasing functions $f(n) \geq 2n$, $\textsc{Time}(f(n)) \subsetneq \textsc{Time}((f(2n+1)^3)$.

Because: there exists a problem HALTING_f decidable in f(2n+1)^3, while HALTING_f' is not decidable in f'(\lfloor \frac{n}{2} \rfloor) = f(n)

so some problem in f(2n+1)^3 is not decidable in f(n) => proven.

注: $\Rightarrow$ $f'(n)$ 不是 $f(n)$ 的导数.

它就是一个新函数. 并且定义:

for $\forall n$, $f(n) = f(2n+1)$.

$\Rightarrow$ 故显然有 $f(n) < f(\lfloor n/2 \rfloor)$.

别问为什么:

- If $f(n)$ is time-constructible and reasonably fast-growing (say $f(n) > 4n$), we *can* decide HALTING$_f$ in time $(f(n))^3$ using a version, U_f of the universal Turing machine:
  - write $f(|x|)$ symbols $\star$ on an 'alarm-clock' (work)tape;
  - simulate the steps of M in the usual way, advancing a counter on the alarm clock tape by 1 for each step;
  - abandon the computation if the alarm clock rings, and just output No.
- U_f can be made to run in time $O(f(n)^3)$, and so can be sped up to run in time $f(n)^3$.

Thm 4:

PTime $\subsetneq$ ExpTime.

证明: 由 Thm 3:

$\because$ PTime $\subseteq$ Time(2^n)

且 $f(n) = 2^n >> 2n$ $\Rightarrow$ Time($f(n)$) $\subsetneq$ Time($f(2n+1)^3$) = Time($2^{3\cdot(2n+1)}$) $\subseteq$ ExpTime. //