作业反馈3-13

TC 26,2,10， 26,2,13 26.1（b） 26.2

26.2-10

Show how to find a maximum flow in a network $G = (V, E)$ by a sequence of at most $|E|$ augmenting paths. (*Hint:* Determine the paths *after* finding the maximum flow.)

先跑一遍最大流算法，得到最大流情况下的流函数 g. 然后，以 g 为 capacity，当迭代地路径增强，使得在用路劲 p 增强使得对 p 上的某条边 (u, v) 有 $f(u, v) = g(u, v)$ 的时候，把 (u, v) 和 (v, u) 从 G_f 中去除 (反正之后不会再改它们了). 由于每次迭代至少使得一条边被这样处理，所以最多进行 $|E|$ 次迭代.

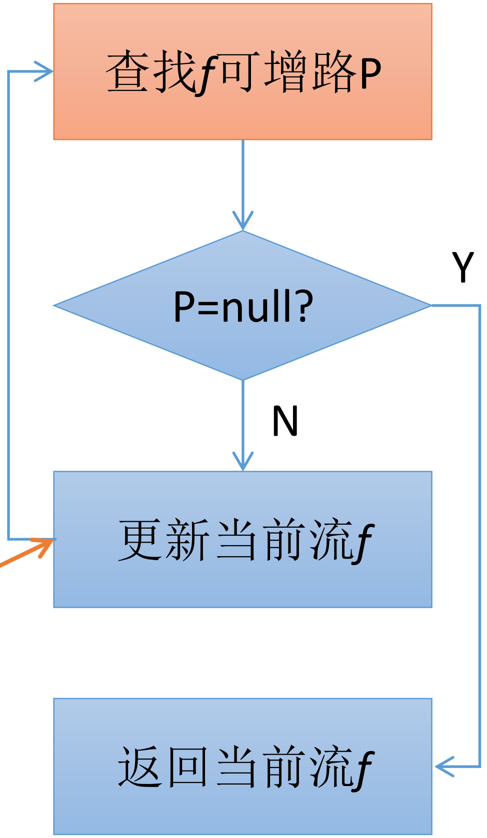
能否能在查找最大流的同时，直接找到这些paths？

Ford-Fulkerson
Edmonds-Karp
都不行

求最大流算法

- 基本思路
  - 反复查找是否存在f可增路P
    - 如果存在,沿P则更新当前流f
    - 否则f即为最大流

$$\hat{f}(a)=\begin{cases} f(a)+\Delta f(P), a\text{为}P\text{的正向弧} \\ f(a)-\Delta f(P), a\text{为}P\text{的反向弧} \\ f(a), \quad\quad a\text{不在}P\text{上} \end{cases}$$



Ford-Fulkerson标号算法

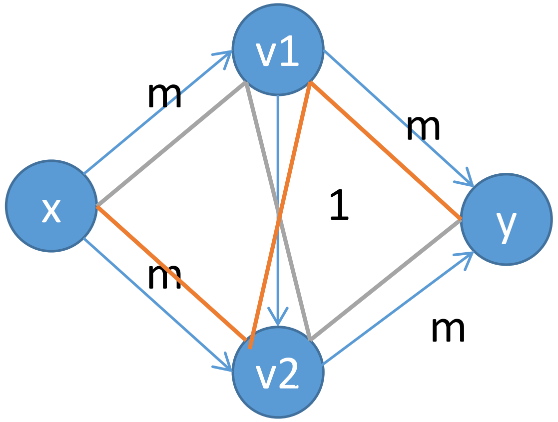
$l(v)$实际表示的是x经过***当前所查找的一条路P***到达v可能的最大流量

- 给定网络N,求N的一个最大流
- 0.初始化: $\forall a \in A$,令$f(a) = 0$;//初始流量为0
- 1. $l(x) = \infty; L = \{x\}; S = \emptyset$//$L$表示已标未查集,$S$表示已标已查集
- 2.while($L \neq \emptyset$)
  - 从L中**任意**取一个节点u, 对所有$v \in N(u) - (L \cup S)$:
    - 如果$a = \langle u, v\rangle \in A$且$c(a) > f(a)$,则给$v$标号:
      $l(v) = \min\{l(u), c(a) - f(a)\}; L = L + \{v\}$
      $(u, +, l(v))$
    - 如果$a = \langle v, u\rangle \in A$且$f(a) > 0$,则给$v$标号:
      $l(v) = \min\{l(u), f(a)\}, L = L + \{v\}$
      $(u, -, l(v))$
  - $L = L - \{u\}; S = S + \{u\}$//u处理完毕
  - if $y \in L$，则存在f可增路, break:
- 3. if $y \in L$(存在f可增路)
    - 顺延标号更新流f
    - **转第1步**
- 4.if $L = \emptyset$不存在f可增路;流f为最大流且 $(S, \bar{S})$为最小割, 返回f

更新f:
1. 令z=y
2. 如果z的标号为$(w, +, l(z))$,令$f(\langle w, z\rangle) = f(\langle w, z\rangle) + l(y)$; 如果z的标号为$(w, -, l(z))$,令$f(\langle z, w\rangle) = f(\langle z, w\rangle) - l(y)$;
3. 如果$w \neq x$,令$z = w$并转2;

Ford-Fulkerson标号算法

- 复杂度$O(\varepsilon v c_{max})$
  - 不仅依赖于问题规模ε, v,还依赖一个参数c_{max}
  - 形式上是ε, v的多项式,但还依赖其他参数——伪多项式



反复查找是否存在*f*可增路*P*
如果存在则更新当前流*f*
否则*f*即为最大流

Edmonds-Karp算法

广度优先遍历,
找最短f可增路

- 给定网络N,求N的一个最大流
- 0.初始化: $\forall a \in A$,令$f(a) = 0$;//初始流量为0
- 1. $l(x) = \infty$; $L = \{x\}$; $S = \emptyset$//L表示已标未查集队列,S表示已标已查集

$O(\varepsilon^2 v)$

- 2.while($L \neq \emptyset$)
  - 从L取第一个节点u, 对所有$v \in N(u) - (L \cup S)$：
    - 如果$a = \langle u, v\rangle \in A$且$c(a) > f(a)$,则给$v$标号：$l(v) = \min\{l(u), c(a) - f(a)\}$; $L = L + \{v\}$
    - 如果$a = \langle v, u\rangle \in A$且$f(a) > 0$,则给$v$标号：$l(v) = \min\{l(u), f(a)\}$, $L = L + \{v\}$
  - $L = L - \{u\}$; $S = S + \{u\}$//u处理完毕
  - if $y \in L$，则存在f可增路, 进行以下操作：
    - 顺延标号更新流f
    - **转第1步**

$\boxed{(u, +, l(v))}$

$\boxed{(u, -, l(v))}$

- 4. $L = \emptyset$不存在f可增路;流f为最大流且$(S, \bar{S})$为最小割, 返回f

更新f:
1. 令z=y
2. 如果z的标号为$(w, +, l(z))$,令$f(\langle w, z\rangle) = f(\langle w, z\rangle) + l(y)$;
如果z的标号为$(w, -, l(z))$,令$f(\langle z, w\rangle) = f(\langle z, w\rangle) - l(y)$;
3. 如果$w \neq x$,令$z = w$并转2;

26.2-13

Suppose that you wish to find, among all minimum cuts in a flow network G with integral capacities, one that contains the smallest number of edges. Show how to modify the capacities of G to create a new flow network G' in which any minimum cut in G' is a minimum cut with the smallest number of edges in G.

称 G' 的 capacity 函数为 c'. 对于 G 中任意一条边 (u, v), 令 $c'(y, v) = c(u, v) + \frac{1}{2|E|}$. 于是, G' 中的最小割的流相比于 G 中的增幅不超过 $1/2$, 对应到 G 中也是一个最小割, 因为要成为一个稍大的割流需要增加不小于 1. 而且, 由于含有边数更多的割受到的增幅更大, G' 中的最小割对应到 G 中一定是边数最少的最小割.

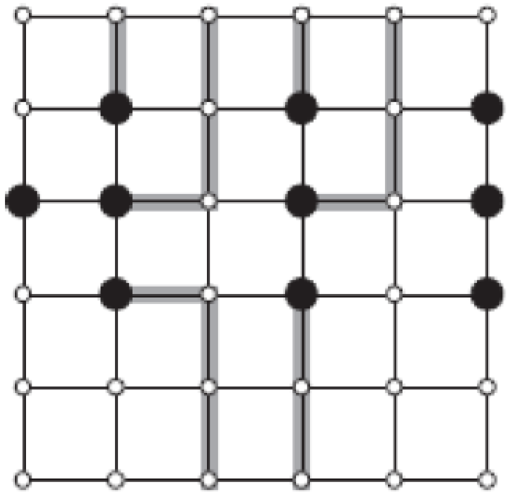
Increment the capacity every edge in G by 1. ?

26-1 *Escape problem*

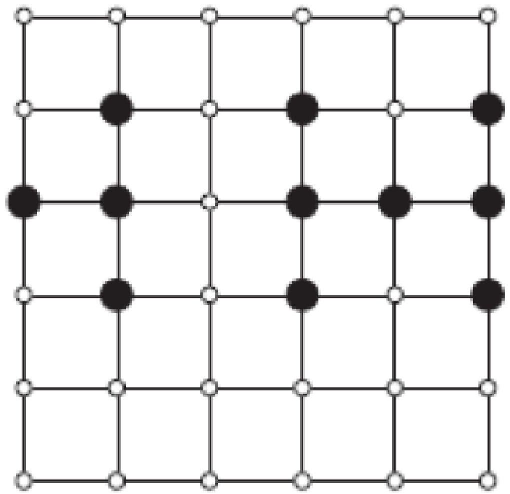
ؤلب

An $n \times n$ ***grid*** is an undirected graph consisting of n rows and n columns of vertices, as shown in Figure 26.11. We denote the vertex in the ith row and the jth column by (i, j). All vertices in a grid have exactly four neighbors, except for the boundary vertices, which are the points (i, j) for which $i = 1$, $i = n$, $j = 1$, or $j = n$.

Given $m \leq n^2$ starting points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ in the grid, the ***escape problem*** is to determine whether or not there are m vertex-disjoint paths from the starting points to any m different points on the boundary. For example, the grid in Figure 26.11(a) has an escape, but the grid in Figure 26.11(b) does not.



(a)



(b)

a. Consider a flow network in which vertices, as well as edges, have capacities. That is, the total positive flow entering any given vertex is subject to a capacity constraint. Show that determining the maximum flow in a network with edge and vertex capacities can be reduced to an ordinary maximum-flow problem on a flow network of comparable size.

26-2 *Minimum path cover*

A ***path cover*** of a directed graph $G = (V, E)$ is a set P of vertex-disjoint paths such that every vertex in V is included in exactly one path in P. Paths may start and end anywhere, and they may be of any length, including 0. A ***minimum path cover*** of G is a path cover containing the fewest possible paths.

a. Give an efficient algorithm to find a minimum path cover of a directed acyclic graph $G = (V, E)$. (*Hint:* Assuming that $V = \{1, 2, \ldots, n\}$, construct the graph $G' = (V', E')$, where

$$\begin{aligned} V' &= \{x_0, x_1, \ldots, x_n\} \cup \{y_0, y_1, \ldots, y_n\} , \\ E' &= \{(x_0, x_i) : i \in V\} \cup \{(y_i, y_0) : i \in V\} \cup \{(x_i, y_j) : (i, j) \in E\} , \end{aligned}$$

and run a maximum-flow algorithm.)

b. Does your algorithm work for directed graphs that contain cycles? Explain.

(a) 首先, 按 Hint 中的方法构造 G', 并把每一条边的 capacity 赋为 1. 于是, 通过最大流算法, 可以得到一些流为 1 的边. 用这些边首位连接来构造最小路径覆盖.

首先, 在构造出来的 walk 中, 没有点会出现两次. 这是因为 G acyclic, 而构造出来的 walk 的边一定出现在 G 中. 然后, 由于连接起来的几个点是一个 path, 单个的点也是一个 path, 所以必然得到一个路径覆盖.

接着说明得到的路径覆盖是最小的. 在 n 个点上, k 条路径的覆盖一共有 $n-k$ 条边. 设由最大流算法构造出的路径覆盖有 k 条路径. 于是, $|f|_M = n-k$. 假设存在一个更小的路径覆盖. 则边数一共有超过 $n-k$ 条. 将它变成对应的流 (一定可以, 因为 path 没有重复点), 则得到 $|f^*| > n-l = |f|_M$. 这显然是不可能的. 因此, 用这个办法得到的是最小的路径覆盖. 若最大流使用 Edmonds-Karp 算法, 则时间复杂度为 $O(|V| \cdot |E|^2)$.

Procedure construct_minimum_vertex_cover

```
P = EMPTY
Repeat until P covers every vertex v ∈ V
    If v ∈ V && v∉P && v' ∉ M
    //Construct a new path and augment this path
        Create an empty path p
        Add v to p
        While v is matched to some vertex w'
            Add w to p
            v=w
        Add p to P
return
```