Algorithm

Homework 4

駱皓正
r05227124
Dept. of Psy

1. (25) Decide whether you think each of the following statements is or false. If it is true, give a short explanation. If it is false, give a counter example.
   A. Let G be an arbitrary flow network, with a source s, a sink t, and a positive integer capacity c_e on every edge e. If f is a maximum s-t flow in G, then f saturates every edge out of s with flow (i.e., for all edges e out of s, we have $f(e)=c_e$).
   B. Let G be an arbitrary flow network, with a source s, a sink t, and a positive integer capacity c_e on every edge e; let (A, B) be a minimum s-t cut with respect to these capacities $\{c_e: e \in E\}$. Now suppose we add 1 to every capacity; then (A, B) is still a minimum s-t cut with respect to new capacities $\{1+c_e: e \in E\}$.

Answer:

For question A:

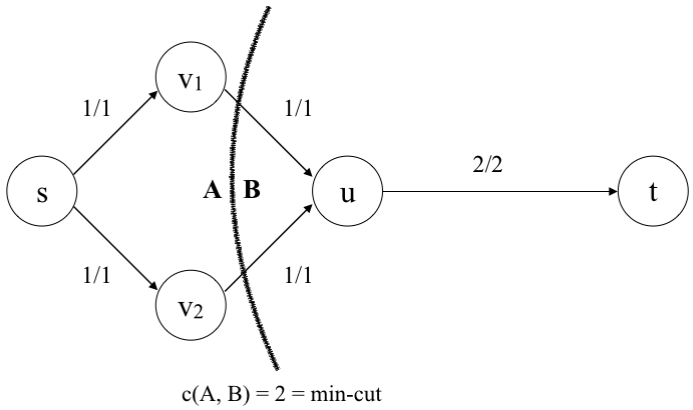
false

Here provides a counter example graph G in which there exists four vertexes a (source), b, c, and d (sink), and edges (a, b), (b, c), and (c, d), and their capacities $c_{(a, b)} = 2$, $c_{(b, c)} = 1$, and $c_{(c, d)} = 2$. The maximum flow f is 1. Because $1 = f((a, b)) \neq c_{(a, b)} = 2$, f doesn't saturate every edge out of a (source) with flow.

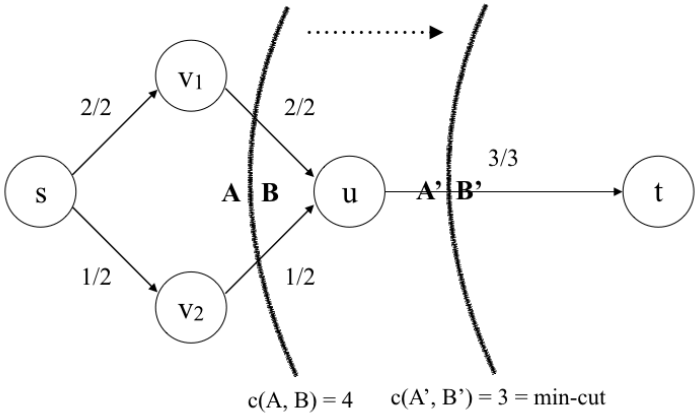
For question B:

false

Here provides a counter example graph G in which there exists five vertexes and five edges and their corresponding flow/capacity showing below:



In *G*, we can find min-cut = 2 = c(A, B). After adding 1 on capacity of each edge in *G*, we get *G'* showing below. We can find that c(A, B) = 4 is no longer the min-cut in *G'*, but the c(A', B') = 3 is. Hence, (A, B) is *not* still a minimum *s-t* cut with respect to new capacities $\{1+c_e: e \in E\}$.



Reference:

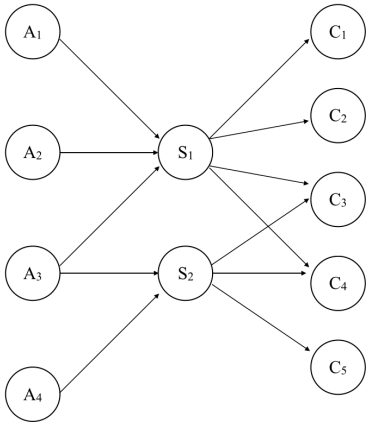
class "Network flow" power point; no coworkers

2. (25) In this year of great depression, a realtor needs to maximize the number of apartments sold; otherwise, she will soon go into bankruptcy. She has p apartments to sell and q potential customers for these apartments. She has m salesmen working for her. Each salesman is assigned a list of apartments and clients interested in these apartments. A salesman can sell an apartment to any of his customers. Salesman i can sell at most b_i apartments. Also, any apartment cannot be owned by more than one person. For $m = 2$, $p = 4$, $q = 5$, $b_1 = b_2 = 2$, and the following assignments of customers and apartments to the salesmen, construct the flow network for the underlying problem. **How to find** the maximum number of apartments that can be sold?

Salesman	Customers	Apartments
1	1,2,3,4	1,2,3
2	3,4,5	3,4

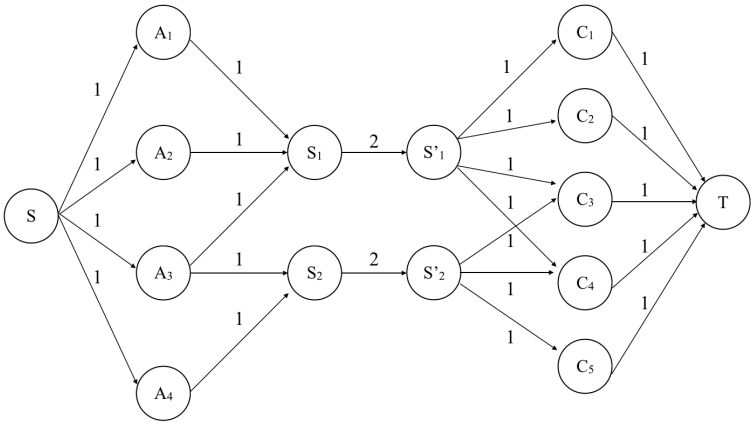
Answer:

Recalling what have mentioned in class Network flow is that network flow always can be employed to solve the problem of "resource distribution." This problem can be seen as distributing the resources (apartments) to the requesters (customers). By the table provided in the problem, I here draw a graph G below to illustrate the Apartments (notated as vertex A_i)—Salesmen (notated as vertex S_i) relation and Salesmen (notated as vertex S_i) — Customers (notated as vertex C_i) relation. The direction of edges indicates apartments are sold to customers via salesmen.

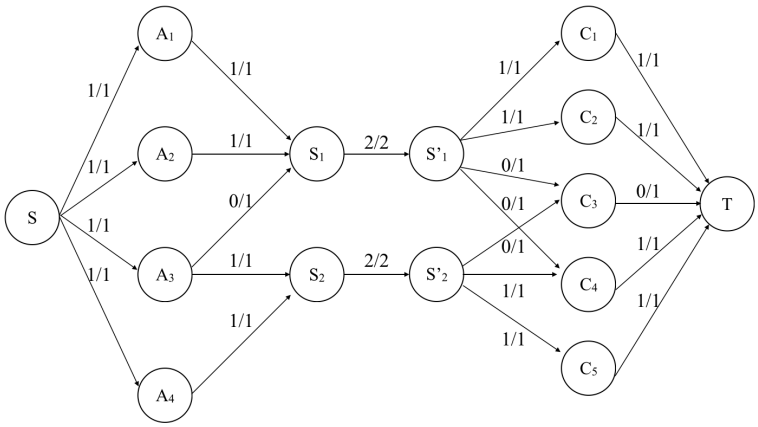


In order to convert the current problem to network flow problem, I build a G' below like bipartite matching. In G', source (notated as vertex S) and sink (notated as vertex T) are attached to original G. In G', I divide S_i into S_i and S'_i, they represent the salesman

who possesses the apartments, and the salesman who possesses the customers respectively. The capacities of (S_1, S'_1) and (S_2, S'_2) are endowed with 2, which indicates the selling ABILITY of each salesman (i.e. only 2 apartments of all apartments belonging to a certain salesman can be sold). In other words, the ability of selling apartments to customers of a salesman is 2, which can be seen as the constraint of between S_i and S'_i. The capacities of other edges are endowed with 1, which indicates the one to one corresponding relationship.



Here we can employ the FORD-FULKERSON algorithm to find the max-flow *f*. The *f* is the maximum number of apartments. The graph *G"* below is one of outcomes (a pattern of resource distribution) of employing FORD-FULKERSON to *G'*. And the *f* is 4.



Reference:

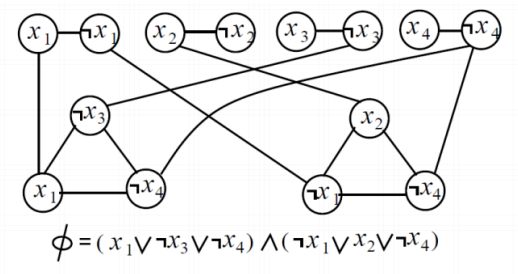
class "Network flow" power point; no coworkers

3. (25) A *vertex cover* of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$ then u or $v \in V'$ (or both). The *Vertex-Cover Problem* (VC) is to find a vertex cover of minimum size in G.

   A. Formally describe the decision problem of VC. Let it be VC_D.

   B. The 3-CNF-SAT problem (3SAT) is defined as follows:

      Definition: 3SAT = {<ϕ>: Boolean formula ϕ in 3-conjunctive normal form is satisfiable}. Use the reduction from 3SAT shown below to prove that VC_D is NP-complete. No partial credit for reduction from any other NP-complete problem. (Hint: What is the size of the vertex cover for the reduced graph?)



Answer:

For question A:

Decision problem formulation: Given an undirected graph $G = (V, E)$ and a bound k, does there exist a vertex cover of size at most k?

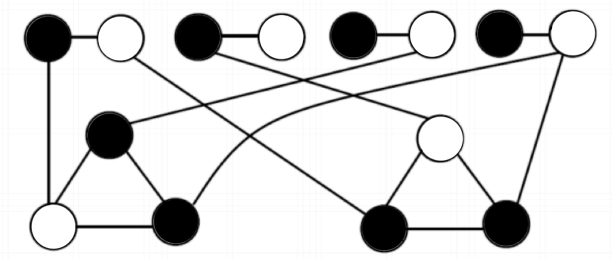
For question B:

- *The 1st step*: **Checking $VC_D \in$ NP:**
  - //need to check a solution in polynomial time
  - Guess a certificate.
  - Check if $V' \subseteq V$.
  - Check if vertex cover of size $\leq k$.
  - //All can be done at $O(n)$, so $VC_D \in$ NP.
- *The 2nd step:* **Checking L' $\leq_p$ VC_D, for every L' $\in$ NP:**
  - //by checking 3SAT $\leq_p$ VC_D, (known $\leq_p$ unknown)
  - *2-1 step:* **find Mapping function 3SAT to VC_D.**
    - Given an instance ϕ of 3SAT with n literals and m clauses.
    - Create an instance (G, k) of a vertex cover of G with $2n+3m$ vertexes with bound k.
    - We can map ϕ to (G, k) by the method in which every literal in ϕ can be mapped into an edge of two vertexes in G, which one would be marked

as the literal and the other would be marked as its negation. In addition, every clause in ϕ can be mapped into a clique of three vertexes in *G*. Here provide a mapping function *f*.

$$f(\phi) = \begin{cases} \text{edge of } u \text{ in } G, \text{if } u \text{ and its negation in } \phi \\ \text{a clique of three vertexes in } G, \text{if a clause in } \phi \end{cases}$$

- We can easily to find mapping will take polynomial time to be done by making edges by literals and three vertexes cycle by clauses.
- Here I set $k = n+2m$, by observing the converted *G* showing below form the example form the question.



➢ *2-2 step:* **ϕ has a 3SAT(satisfiable) if and only if the reduced instance has a VC_D with size ≤ *k*.**

- //ϕ ∈3SAT → *f*(ϕ) ∈ VC_D
- If 3SAT is satisfiable, then the assignment is legal to makes every clause be true. Here, we can choose the vertexes corresponding to the TRUE literal of ϕ in 3SAT to be in vertex cover. Because in each clause existing at least one TRUE literal, for covering the rest of edges in the clique, we can choose at most the other two vertexes corresponding to the other two variables in clause to be in vertex cover. Hence, if ϕ is a true instance, then we can find *f*(ϕ) has a VC_D with size ≤ $k = n + 2m$.

➢ *2-3 step:* **ϕ has a 3SAT(satisfiable) if and only if the reduced instance has a VC_D with size ≤ *k*.**

- //*f*(ϕ) ∈ VC_D → ϕ ∈3SAT
- If exists a vertex cover of size being equal to or less than $n + 2m$, we can assure that at least of n vertexes can correspondingly cover the whole literal part because it cannot be cover by other means. And the rest of $2m$ vertexes can cover the clauses. This ϕ would be true, since we can take literals corresponding to the vertex cover to be our assignment. And this assignment must satisfy the formula since every clause is true.

➢ Hence, 3SAT ≤$_p$ VC_D

Thus, by the proof above, we can say VC_D is a NP-complete problem.

Reference:
class "NP-complete" power point;
video of NP-complete problem by UHMICSAlogithms
https://www.youtube.com/watch?v=J5l-crl0LgA
no coworkers

4. (25) Assume you are creating an array data structure that has a fixed size of n. You want to backup this array after every so many insertion operations. Unfortunately, the backup operation is quite expensive, it takes n time to do the backup. Insertions without a backup just take 1 time unit.
   A. How frequently can you do a backup and still guarantee that the amortized cost of insertion is O(1)?
   B. Prove that you can do backups in O(1) amortized time by using the potential method. (Hint: Let $\phi_i = i \bmod n$.)

Answer:

For question A:

We can do back up the array right after every n insertion.

For question B:

proof: by potential method

$$\text{Let } \phi_i = i \bmod n$$

To find the amortized cost of an insertion, here split into two cases. In first case, suppose backing up right after n insertion, so $i = kn$, which $k \in \mathbb{N}$. That is $i \bmod n = 0$. Here the actual cost = n, so the amortized cost of insertion is:

$$\widehat{c_i} = c_i + \phi_i - \phi_{i-1} = n + (n \bmod n) - (n - 1 \bmod n) = n + 0 - n + 1 = 1$$

In the second case, suppose we need no backing up after insertion. That is $i \bmod n \neq 0$. The amortized cost is:

$$\widehat{c_i} = c_i + \phi_i - \phi_{i-1} = 1 + (i \bmod n) - (i - 1 \bmod n) = 1 + i - i + 1 = 2$$

Thus, we can do backups in O(1) amortized time.

Reference:

class "amortize analysis" power point;

no coworkers