Lecture 2

Fast Cut algo

review **Contrast Algorithm** for Min-Cut problem

```
1 Contract Algo:
2     Pick an edge uniformly at random;
3     Merge the endpoints of this edge;
4     Remove self-loops;
5     Repeat steps in line 2-4 until only two vertices left;
6     The remaining edges form a candidata cut;
```

time complexity: $O(n^2)$
successful probability: $\Omega(\frac{1}{n^2})$

improvement:

```
Fast Cut algorithm: ref Randomized Algo 10.2
Input: multigraph G(V,E)
Output: cut C
    if n <= 6:
         compute min-cut by brute-force enumeration
    else:
        t = [1+n/sqrt2]
        Using Contract algo, perform two independent contract sequences to obtain graphs H1 and H2
        ...each with t vertices
        Recursively compute min-cuts in each of H1,H2
        return the smaller of the two min-cuts

```

Time complexity:

$$T(n) = 2T(\frac{n}{\sqrt{2}}) + O(n^2)$$

$$T(n) = O(n^2 \log n)$$

Successful probablity:
the origin size n problem is reduced to two subproblem size $\frac{n}{\sqrt{2}}$, considering the contract procedure, the probability that the result of subproblem is exactly the result of origin problem is

$$(1-\frac{2}{n})(1-\frac{2}{n-1})...(1-\frac{2}{n/\sqrt{2}})=\frac{(\frac{n}{\sqrt{2}}-1)(\frac{n}{\sqrt{2}}-2)}{n(n-1)}\geq\frac{1}{2}$$

denote the successful probabilty of origin problem with size n by $P(k+1)$, subproblem with size $\frac{n}{\sqrt{2}}$ by $P(k)$, then we have

$$P(k+1)=1-(1-\frac{1}{2}P(k))^2$$

so we get

$$P(k+1)=P(k)-\frac{1}{4}P(k)^2$$

perform a change of variable $P(k)=\frac{4}{q(k)+1}$, yields the following simplification:

$$q(k+1)=q(k)+1+\frac{1}{q(k)}$$

recursively apply the equation to the righthand side

$$q(k)=q(1)+k-1+\sum_{1\leq i\leq k-1}\frac{1}{q(k)}\geq k-1$$

by using $q(k)\geq k-1$,we have

$$q(k)\leq q(1)+k-1+\sum_{1\leq i\leq k-1}\frac{1}{i}\leq q(1)+k-1+\log k$$

combine the upper bound and lower bound

$$q(k)=\Theta(k)$$

$$P(k)=\Theta(\frac{1}{k})$$

observe that $k=\log n$,we can draw the conclution

$$Pr(n) = \Theta(\frac{1}{\log n})$$

Las Vegas VS. Monte Carlo

- Las Veags: random running time (quick sort)
- Monte Carlo:randomized quality of solution(randomized min cut)

Complexity Class

only consider decision problem in this chapter

P and NP

Definition(P):

$$L \in P \iff \exists \text{Polynomial time algorithm A} s.t. \forall x \in \Sigma^*, \begin{cases} x \in L \Rightarrow A(x) accepts \\ x \notin L \Rightarrow A(x) rejects \end{cases}$$

Definition(NP):

$$L \in NP \iff \exists \text{Polynomial time algorithm A} s.t. \forall x \in \Sigma^*, \begin{cases} x \in L \Rightarrow \exists y, |y| = poly|x|, A(x,y) accepts \\ x \notin L \Rightarrow \forall y, |y| = poly|x|, A(x,y) rejects \end{cases}$$

NP-hard problem(informal definition):A is NP-hard ⇔ if A is polynomial time solvable, all problems in NP are polynomial time solvable

NP-complete:both NP-hard and in NP

famous NP-complete problems: 3-SAT,Vertex cover,Set cover,Clique,Independent set,integer programming...

RP CoRP

Definition(RP randomized poly time):

$$L \in RP \iff \exists \text{randomized algorithm A running in worst-case polynomial time} s.t. \forall x \in \Sigma^*, \begin{cases} x \in L \Rightarrow Pr(A(x) accepts) \geq \frac{1}{2} \\ x \notin L \Rightarrow Pr(A(x) accepts) = 0 \end{cases}$$

Definition(RP):

$$L \in co-RP \iff$$

$\exists \text{randomized algorithm A running in worst-case polynomial time} s.t. \forall x \in \Sigma^*, \begin{cases} x \in L \Rightarrow Pr(A(x) accepts) = 1 \\ x \notin L \Rightarrow Pr(A(x) accepts) \leq \frac{1}{2} \end{cases}$

RP and Co-RP are defined with one side-error,which means that,for $L \in RP$ and randomized algorithm A, if A say "$x \in L$",it is definitely correct,but if A say not,it could be wrong.

Theorem:
(1)$RP \subseteq NP$
(2)$coRP \subseteq coNP$
Prove (1):
consider randomized algorithm A,need to notice that if we know how A produces random number,than we can regard A as a deterministic algorithm.
denote the random string of A is r,for any x,$|r| \leq poly|x|$,construct algorithm B(x,r)=A(x)

BPP,PP,ZPP

These three complexity classes deal with *error*
Definition(BPP bounded error prob poly time):
$L \in BPP \iff$
$\exists \text{randomized algorithm A running in worst-case polynomial time} s.t. \forall x \in \Sigma^*, \begin{cases} x \in L \Rightarrow Pr(A(x) accepts) \geq \frac{3}{4} \\ x \notin L \Rightarrow Pr(A(x) accepts) \leq \frac{1}{4} \end{cases}$

Definition(PP prob poly time):
$L \in BPP \iff$
$\exists \text{randomized algorithm A running in worst-case polynomial time} s.t. \forall x \in \Sigma^*, \begin{cases} x \in L \Rightarrow Pr(A(x) accepts) > \frac{1}{2} \\ x \notin L \Rightarrow Pr(A(x) accepts) < \frac{1}{2} \end{cases}$

Definition(ZPP Zero-error prob poly time):
the class of language that have **Las Vegas** algorithm running in **expected** polynomial time

Theorem:

1. $RP \subseteq BPP (\subseteq PP)$
2. $NP \subseteq PP$
3. $ZPP = RP \cap coRP$

Prove:
(1)repeat
(2)NP:algorithm A
PP:algorithm B:randomly choose y as witness,|y|=poly|x|

if A(x,y) accepts, B accept
if A(x,y) rejects, B has probabilty 1/2 to accept, 1/2 to reject
(3)ZPP algo A; RP,coRP algo B1,B2
"$\subseteq$" by using "cut off"
given $|x|^n$ as time limit
B1&B2: execute A(x) in $|x|^n$ time,if A accepts\rejects, B accepts\rejects,else if A does not stop,B1 rejects,B2 accepts.

Markov inequality:

$$Pr(X \geq c) \leq \frac{E(x)}{c}$$

guarantees the probabilty bound.
"$\supseteq$"

```
1 ZPP algorithm A:
2 Input:x
3     execute B1(x),if B1 accepts,A accepts
4     execute B2(x),if B2 rejects,B rejects
5     repeat line 3-4
```

repeat 3-4 one time has probabilty 3/4 to stop,which ensures the expected running time is poly.

Relations

$$P \subseteq RP \subseteq^{NP}_{BPP} \subseteq PP \subseteq PSPACE$$