Home Assignment 2
Advanced Algorithms and Data Structure

Siyi Wu	Zhongxing Ren	Yu Pei	Qianxi Yang
KRW521	CPJ395	MQH875	HSB264

December 6, 2022

1 Group Part

1.1 Answer of exercise 29.1-5 in CLRS

1. **original linear program**

$$
\begin{aligned}
\textbf{maximize} \quad & 2x_1 - 6x_3 \\
\textbf{subject to} \quad & \\
& x_1 + x_2 - x_3 \leq 7 \\
& 3x_1 - x_2 \geq 8 \\
& -x_1 + 2x_2 + 3x_3 \geq 0 \\
& x_1, x_2, x_3 \geq 0
\end{aligned}
$$

2. **slack form**

$$
\begin{aligned}
z &= 2x_1 - 6x_3 \\
x_4 &= 7 - x_1 - x_2 + x_3 \\
x_5 &= -8 + 3x_1 - x_2 \\
x_6 &= -x_1 + 2x_2 + 3x_3 \\
& x_1, x_2, x_3, x_4, x_5, x_6 \geq 0
\end{aligned}
$$

according to the definition of slack form, x_1, x_2, x_3 are basic variables and x_4, x_5, x_6 are nonbasic form.

1.2 Answer of exercise 29.2-6 in CLRS

For a bipartite graph G = (V,E), the vertices set V can be divided into two disjoint sets of vertices V_1, V_2. And for any edge $e =< i, j >\in E$, if $i \in V_1$, then $j \in V_2$.

Then we can write a linear program to find the maximal matching of graph G. For every edges $< i, j >$, a new variable X_{ij} is introduced. If the edge $< i, j >$ is in the matching set, $X_{ij} = 1$, else $X_{ij} = 0$.

Multiple edges with the same vertex have at most one edge in the matching set, so the maximum of the set is:

$S = \sum_{e \in E} X_{ij}$

Subject to constraints:
for the vertex i: $X_{i..} + X_{i..} \leq 1$
for the vertex j: $X_{..j} + X_{..j} \leq 1$
$X_{ij} \geq 0$

While $X_{i..}$ means an edge with the vertex i and $X_{..j}$ means an edge with

the vertex j.

1.3 Answer of exercise 29.3-5 in CLRS

$$
\begin{array}{lllll}
\text{maximize} & 18x_1 & +12.5x_2 & & \\
\text{subject to} & x_1 & +x_2 & & \leq 20 \\
& x_1 & & & \leq 12 \\
& & x_2 & & \leq 16 \\
& x_1, & x_2 & & \geq 0
\end{array}
$$

Step 1:

$$
\begin{array}{lllllll}
\text{maximize} & 18x_1 & +12.5x_2 & & & & \\
\text{subject to} & x_1 & +x_2 & +x_3 & & & = 20 \\
& x_1 & & & +x_4 & & = 12 \\
& & x_2 & & & +x_5 & = 16 \\
& x_1, & x_2, & x_3, & x_4, & x_5 & \geq 0
\end{array}
$$

Step 2:

$$
\begin{aligned}
z &= 0 + 18x_1 + 12.5x_2 \\
x_3 &= 20 - x_1 - x_2 \\
x_4 &= 12 - x_1 \\
x_5 &= 16 - x_2
\end{aligned}
$$

Step 3:
If x_1 is increased beyond 20 then x_3 becomes negative.
If x_1 is increased beyond 12 then x_4 becomes negative.
If x_1 is increased then x_5 remains unchanged.
Constraint defining x_4 is binding.

$$
\begin{aligned}
z &= 0 + 18(12 - x_4) + 12.5x_2 \\
x_3 &= 20 - (12 - x_4) - x_2 \\
x_1 &= 12 - x_4 \\
x_5 &= 16 - x_2
\end{aligned}
$$

$$
\begin{aligned}
z &= 216 + 12.5x_2 - 18x_4 \\
x_3 &= 8 - x_2 + x_4 \\
x_1 &= 12 - x_4 \\
x_5 &= 16 - x_2
\end{aligned}
$$

New basic variables: $x_1 = 12, x_3 = 8, x_5 = 16$
New objective value: $z = 216$
New feasible basic solution: $(12, 0, 8, 0, 16)$

Step 4:
If x_2 is increased beyond 16 then x_5 becomes negative.

If x_2 is increased beyond 8 then x_3 becomes negative.
If x_2 is increased then x_1 remains unchanged.
Constraint defining x_3 is binding.

$$
\begin{aligned}
z &= 216 + 12.5(8 - x_3 + x_4) - 18x_4 \\
x_2 &= 8 - x_3 + x_4 \\
x_1 &= 12 - x_4 \\
x_5 &= 16 - (8 - x_3 + x_4)
\end{aligned}
$$

$$
\begin{aligned}
z &= 316 - 12.5x_3 - 5.5x_4 \\
x_2 &= 8 - x_2 + x_4 \\
x_1 &= 12 - x_4 \\
x_5 &= 8 + x_3 - x_4
\end{aligned}
$$

New basic variables: $x_1 = 12, x_2 = 8, x_5 = 8$
New objective value: $z = 316$
New feasible basic solution: $(12, 8, 0, 0, 8)$

Stop, since no more non-basic variables appear in the objective with a positive coefficient.

1.4 Answer of exercise 29.4-1 in CLRS

$$
A = \begin{pmatrix} 0 & 18 & 12.5 \\ 20 & -1 & -1 \\ 12 & -1 & 0 \\ 16 & 0 & -1 \end{pmatrix} \Rightarrow A^T = \begin{pmatrix} 0 & 20 & 12 & 16 \\ 18 & -1 & -1 & 0 \\ 12.5 & -1 & 0 & -1 \end{pmatrix}
$$

After transposing A, now we switch to minimize $20y_1 + 12y_2 + 16y_3$, and the constraints become:

$$
\begin{aligned}
y_1 + y_2 &\geq 18 \\
y_1 + y_3 &\geq 12.5 \\
y_1, y_2, y_3 &\geq 0
\end{aligned}
$$

1.5 Answer of Exercises 1 for Randomized Algorithms

In fact, $d(x) = 1 + h$, where h is the number of ancestors of x and I set the depth of the root is 1. And I assume that the index of x in the sorted S is a (i.e. S(a)=x). Here I define Y_{ia} as "whether $S_{(i)}$ is the ancestor of $S_{(a)}$ or not" $(i < a)$, and Y_{aj} as "whether $S_{(j)}$ is the ancestor of $S_{(a)}$ or not" $(a < j)$, so $Y_{ia}, Y_{aj} \in 0, 1$. Likewise, I define p'_{ia} as "the probability of $S_{(i)}$ is the ancestor of $S_{(a)}$" $(i < a)$, and p'_{aj} as "the probability of $S_{(j)}$ is the ancestor of $S_{(a)}$" $(a < j)$. Like the analysis in the slide, in the sublist returned in each recursive call in **randQS**, for $i < a$, $S_(i)$ will be one of the ancestors of $S_(a)$ if and only if $S_(i)$

is selected as the pivot. for $a < j$, j will be one of the ancestors of $S_(a)$ if and only if $S_(j)$ is selected as the pivot. So $p'_{ia} = \frac{1}{a-i+1}$, $p'_{aj} = \frac{1}{j-a+1}$. So,

$$\begin{aligned}
E[d(x)] &= 1 + E[\sum_{i=1}^{i=a-1} Y_{ia}] + E[\sum_{j=a+1}^{j=n} Y_{aj}] \\
&= 1 + \sum_{i=1}^{i=a-1} E[Y_{ia}] + \sum_{j=a+1}^{j=n} E[Y_{aj}] \\
&= 1 + \sum_{i=1}^{i=a-1} [1 * p'_{ia} + 0 * (1 - p'_{ia})] + \sum_{j=a+1}^{j=n} [1 * p'_{aj} + 0 * (1 - p'_{aj})] \\
&= 1 + \sum_{i=1}^{i=a-1} p'_{ia} + \sum_{j=a+1}^{j=n} p'_{aj} \\
&= 1 + \sum_{i=1}^{i=a-1} \frac{1}{a-i+1} + \sum_{j=a+1}^{j=n} \frac{1}{j-a+1}
\end{aligned}$$

For lower bound:

$$\begin{aligned}
E[d(x)] &\geq \sum_{k=1}^{k=n} \frac{1}{k} \\
&= H_n \\
&> \int_1^{n+1} \frac{1}{x} dx \\
&= ln(n+1)
\end{aligned}$$

For upper bound:

$$\begin{aligned}
E[d(x)] &< 1 + 2(\sum_{k=2}^{k=n} \frac{1}{k}) \\
&= 1 + 2(H_n - 1) \\
&< 1 + 2\int_1^n \frac{1}{x} dx \\
&= 1 + 2ln(n)
\end{aligned}$$

So here we can determine that $f(n) = logn$, and the derivation above can prove that $E[d(x)] = \Theta(logn)$.

1.6 Answer of Exercises 2 for Randomized Algorithms

We know that if we call **RandMinCut(G)** $t\frac{n(n-1)}{2}$ times and get the returned cut, then

$$Pr[\text{not a min-cut}] \leq e^{-t} \tag{1}$$

and that is proved in the slide of the course. So, making sure that Pr[min-cut] $\geq 99\%$ will be Pr[not a min-cut] ≤ 0.01. So $e^{-t} \leq 0.01$, and we can have $t \geq ln100$. So, to make sure that Pr[min-cut] $\geq 99\%$, we need at least $\lceil \frac{n(n-1)}{2} * ln100 \rceil$ runs of **RandMinCut(G)**.

1.7 Answer of Exercises 3 for Randomized Algorithms

1.7.1 Answer of 1.2 in pdf

Here I provide one kind of input graph:

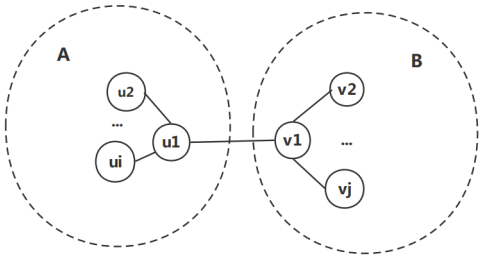

Figure 1: An hourglass-like graph

It is an hourglass-like graph, vertices u_1 to u_i link each other randomly, and vertices v_1 to v_j link each other randomly, but there is only one edge between **u**s and **v**s, that is, (u_1, v_1). Obviously, the min-cut **C** of the graph only contains (u_1, v_1), and $|C| = 1$.
And here I also define a kind of non-empty set named **vertices coalesce set**. In every iteration of the modified algorithm, there will be 0 or 2 vertices in this set selected by the modified algorithm to coalesce until there is only one vertex being left.
So, for the hourglass-like graph, we can get the min-cut if and only if u_1 to u_i are in one **vertices coalesce set** and v_1 to v_j are in another **vertices coalesce set** so that the modified algorithm only choose two vertices in **u**s or two vertices **v**s to coalesce in every iteration and the edge in min-cut can be saved. In fact, the min-cut cannot be found as long as the modified algorithm coalesce **u** vertex and **v** vertex (that also means some **u** vertices and some **v** vertices are in the same **vertices coalesce set**) in one or several iterations, because the small cut found at last will contain two or more edges (and that is not the min-cut we want).
So, to figure out the probability that the modified algorithm finds the min-cut is to compare the number of possible division (into **vertices coalesce set A** and **vertices coalesce set B**).

$$Pr[\text{min-cut}] = \frac{2}{2^n - 2} = \frac{1}{2^{n-1} - 1} \tag{2}$$

The numerator 2 means u_1 to u_i are in one **vertices coalesce set A** (or **B**) and v_1 to v_j are in another **vertices coalesce set B** (or **A**). The denominator $2^n - 2$ means all the possible division except **vertices coalesce set A** is empty or **vertices coalesce set B** is empty.
So there are inputs (the hourglass-like graphs) on which the probability that this modified algorithm finds a min-cut is exponentially small.

1.7.2 Answer of 1.3 in pdf

The Las Vegas algorithm will be like: "A run containing using the Mote Carlo algorithm **A** with time at most T(n) to figure out a result and verifying the correctness of the result given by algorithm **A** in time t(n). After a run, if the result is correct, then the Las Vegas algorithm will return the correct answer and end, otherwise the Las Vegas algorithm will start another run until the result is correct." In one run, the consuming time is at most $T(n) + t(n)$, and the probability of getting a correct result after a run is $\gamma(n)$, so the number of runs (let's say **r**) to get the first correct result after a run follows geometric distribution. So according to the feature of geometry distribution, $E(r) = \frac{1}{\gamma(n)}$. So such Las Vegas algorithm always gives a correct answer to $\sqcap$ and runs in expected time at most $\frac{(T(n)+t(n))}{\gamma(n)}$.

2 Own Summary of Each Group Member

2.1 Own summary of Siyi Wu (KRW521)

2.1.1 Linear Programming

1. General linear programs
2. Standard form:

$$\begin{aligned} &\text{maximize} \\ &\qquad \sum_{j=1}^{n} c_j x_j \\ &\text{subject to} \\ &\qquad \sum_{j=1}^{n} a_{ij} x_j \le b_i \text{ for } i = 1, 2, \ldots, m \\ &\qquad x_j \ge 0 \text{ for } j = 1, 2, \ldots, n \end{aligned}$$

3. Converting linear programs into standard form: 4 situations

4. Slack form:

$$\sum_{j=1}^{n} c_j x_j \Rightarrow z = v + \sum_{j=1}^{n} c_j x_j$$
$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \Rightarrow x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_i$$
$$x_j \geq 0 \text{ for } j = 1, 2, \ldots, n+m$$

5. Converting linear programs into slack form: The simplex algorithm

2.1.2 Randomized Algorithms

1. *RandQS*

$$E[\#\text{comparisons}] = E\left[\sum_{i<j} X_{ij}\right] = \sum_{i<j} E\left[X_{ij}\right] = \sum_{i<j} p_{ij}$$
$$E\left[X_{ij}\right] = (1 - p_{ij}) \cdot 0 + p_{ij} \cdot 1 = p_{ij}$$

a	$\cdots$	i	$\cdots$	j	$\cdots$	b

Suppose a sorted list is divided into three segments a-i, i-j and j-b. a-i and j-b do not affect p_{ij}.
The p_{ij} is calculated by the probability of comparing i with j in i-j segment.

$$p_{ij} = \frac{2}{j+1-i}$$

$$\begin{aligned} E[\#\text{comparisons}] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &= 2 \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{1}{k} \\ &\leq 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{1}{k} \\ &= 2 \sum_{i=1}^{n-1} H_n \\ &< 2nH_n = O(n \log n) \end{aligned}$$

2. *RandMinCut*

$$\Pr[C \text{ is returned }] \geq \frac{2}{n(n-1)}$$

Calling *RandMinCut* $t\frac{n(n-1)}{2}$ times.

$$\begin{aligned}\Pr\left[C^\star \text{ is not a min-cut}\right] &\leq \left(1-\frac{2}{n(n-1)}\right)^{t\frac{n(n-1)}{2}} \\ &\leq \left(e^{-\frac{2}{n(n-1)}}\right)^{t\frac{n(n-1)}{2}} \\ &= e^{-t}\end{aligned}$$

2.2 Own summary of Zhongxing Ren (CPJ395)

2.2.1 Linear Programming

- Start with the example in class, Polly wants to eat in a way that she can meet her requirement of energy, nutrient and limit of her stomach at the least price. And such kind of problem in our life can be abstracted and become the Linear Programming problem.
- In a general **Linear Programming** problem, there is an objective function that is the polynomial we want to maximize or minimize (just like the polynomial of price that we want to minimize in the Polly's diet example), and there are also some linear constraints on the variable (like the requirement of energy, nutrient and limit of her stomach in the Polly's diet example). And such general **Linear Programming** problem can be transformed into a standard form linear programming or a slack form linear programming.
- Standard form linear programming:

$$\begin{aligned}&\text{maximize} \\ &\qquad \sum_{j=1}^{n} c_j x_j \\ &\text{subject to} \\ &\qquad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \text{ for } i = 1, 2, \ldots, m \\ &\qquad x_j \geq 0 \text{ for } j = 1, 2, \ldots, n.\end{aligned}$$

Transforming a general **Linear Programming** problem into a standard form linear programming (several steps): maximize objectives, "=" $\Rightarrow$ " $\leq$, $\geq$ ", making constraints "$\leq$" and all elements (i.e. all x) non-negative.

- Slack form linear programming:

$$\sum_{j=1}^{n} c_j x_j \Rightarrow z = v + \sum_{j=1}^{n} c_j x_j$$
$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \Rightarrow x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_i$$
$$x_j \geq 0 \text{ for } j = 1, 2, \ldots, n+m$$

  Transforming a standard form linear programming into a slack form linear programming (several steps): objective becomes z, x_{n+i}, all elements (i.e. all x (new added included)) non-negative.

- Using SIMPLEX to solve the LP in slack form (Pivoting until all the coefficients of x in z=... become negative, if cannot, then there is no solution).

- dual(for standard form): change max to min, change x to y, change constraint from $\leq$ to $\geq$, transpose the coefficient matrix, exchange the right size number of constraint and coefficient of objective (left to up, right to down)

2.2.2 Randomized Algorithms

- Some algorithms have randomization in them, and such algorithms are called **Randomized Algorithms**.

- RandQS: choose elements of the array(s) randomly in every iteration to split the array(s). To measure the consuming time, we should find out the times we need for comparison.

$$E[X_{ij}] = p_{ij}$$
$$E[\#\text{comparisons}] = E\left[\sum_{i<j} X_{ij}\right] = \sum_{i<j} E[X_{ij}] = \sum_{i<j} p_{ij}$$

  Each iteration returns some sublists $\left[S_{(a)}, \cdots, S_{(b)}\right]$

a	$\cdots$	i	$\cdots$	j	$\cdots$	b

  let $x = S_{(a)}$ become the pivot, if $a < iora > j$,then it doesn't determine whether $S_{(i)}$ and $S_{(j)}$ will compare; if $i < a < j$, then $S_{(i)}$ and $S_{(j)}$ won't compare; if $a = iora = j$, then $S_{(i)}$ and $S_{(j)}$ will compare-¿

$$p_{ij} = \frac{2}{j+1-i}$$

so

$$\begin{aligned} E[\#\text{comparisons}] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &\leq 2nln(n) \\ &= O(n \log n) \end{aligned}$$

- Rand min-cut: In every iteration, we choose the edge to contract randomly,and calculate the probability of getting the min-cut (i.e. continued multiply the probability that we choose the edge that is not in the min-cut to contract in every iteration):

$$|E_i| = \frac{1}{2} \sum_{v \in V_i} d_i(v) \geq \frac{1}{2}|V_i||C|$$

$$\Pr[C \text{ is returned }] \geq \frac{2}{n(n-1)}$$

To increase the probability to get the min-cut, we can call *RandMinCut* $t\frac{n(n-1)}{2}$ times.

$$\Pr\left[C^\star \text{ is not a min-cut}\right] \leq \left(1 - \frac{2}{n(n-1)}\right)^{t\frac{n(n-1)}{2}}$$

$$= e^{-t}$$

So

$$\Pr\left[C^\star \text{ is a min-cut}\right] \geq 1 - e^{-t}$$

2.3 Own summary of Yu Pei (MQH875)

2.3.1 Linear Programming

Linear programming has a lot of applications in real life. A general linear programming problem aims to maximize or minimize the objective function that describes some relationship between the variables. These variables have some linear constraints. In standard form, we always maximize the objective function, and set non-negative constraints on all n variables. Also, every constraint inequality must be <=.

- Standard Form:

$$\begin{aligned} &\text{maximize} \\ &\qquad \sum_{j=1}^{n} c_j x_j \\ &\text{s.t.} \\ &\qquad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \text{ for } i = 1, 2, \ldots, m \\ &\qquad x_j \geq 0 \text{ for } j = 1, 2, \ldots, n. \end{aligned}$$

- Slack Form:

  We introduce slack variables and rewrite the LP to slack form. In slack form, left-hand side variables are called basic variables and right-hand side are called nonbasic. SIMPLEX computes a feasible basic solution by picking a variable in the objective function which positively increases the value and pivoting it with the basic variable that bottlenecks how much the non-basic variable can be increased. SIMPLEX terminates when all coefficients in the objective function become negative or when this LP appears to be unbounded.

- The DUAL of a linear program(standard form):

  change maximize to minimize; transpose the coefficient matrix

$$\begin{aligned} &\text{minimize} \\ &\qquad \sum_{i=1}^{m} b_i y_i \\ &\text{subject to} \\ &\qquad \sum_{i=1}^{m} a_{ij} y_i \geq c_j \text{ for } j = 1, 2, \ldots, n \\ &\qquad y_i \geq 0 \text{ for } i = 1, 2, \ldots, m. \end{aligned}$$

2.3.2 Randomized Algorithms

- RandQS Algorithm:

  we measure the running time in terms of the number of comparisons it performs $S_{(i)}$ and $S_{(j)}$ are compared if and only if $S_{(i)}$ or $S_{(j)}$ is the first pivot among $S_{(i)}, \ldots, S_{(j)}$. There are $j + 1 - i$ elements in this set, and we choose pivot uniformly at random. Therefore, the probability p_{ij} that $S_{(i)}$ and $S_{(j)}$ are compared $\frac{2}{j+1-i}$

$$E[\#\text{comparisons}] = \sum_{i=1}^{n}\sum_{j>i} E[X_{ij}] = \sum_{i=1}^{n}\sum_{j>i} p_{ij} = \sum_{i=1}^{n}\sum_{j>i} \frac{2}{j-i+1}$$
$$= \sum_{i=1}^{n}\sum_{k=2}^{n-i+1} \frac{2}{k} \leq \sum_{i=1}^{n}\sum_{k=1}^{n} \frac{2}{k} = 2n\sum_{k=1}^{n} \frac{1}{k} = O(nlogn)$$

- Min-Cut Algorithm For any min-cut C,

$$\Pr[\textit{RandMinCut} \text{ returns } C] \geq \frac{2}{n(n-1)}$$

To increase the probability of getting the min-cut, we can run *RandMinCut* $x := t\frac{n(n-1)}{2}$ times.

$$\begin{aligned}\Pr\left[\text{not finding min-cut}\right] &\leq (1-\frac{t}{x})^x \\ &\leq (e^{-\frac{t}{x}})^x \\ &= e^{-t}\end{aligned}$$

2.4 Own summary of Qianxi Yang (HSB264)

2.4.1 Linear programming

Linear programming is a process that is used to determine the best outcome of a linear function and is known as the objective function. Usually, linear function or objective function consists of linear equality and inequality constraints. We obtain the best outcome by minimizing or maximizing the objective function. The most important part of solving linear programming problems is to first formulate the problem using the given data.

The standard form of a linear programming problem is given below:

1. Maximization of a linear function
2. n non-negative real-valued variables
3. m linear inequalities("less than or equal to")

$$\begin{aligned}&\text{maximize} \\ &\qquad \sum_{j=1}^{n} c_j x_j \\ &\text{s.t.} \\ &\qquad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \text{ for } i = 1, 2, \ldots, m \\ &\qquad x_j \geq 0 \text{ for } j = 1, 2, \ldots, n.\end{aligned}$$

We next rewrite the LP into slack form. It is a more convenient form for describing the Simplex algorithm for solving LP.

Specifically, we can rewrite LP, so that every inequality becomes equality, and all variables must be positive; namely, the new LP will have a form depicted on the right (using matrix notation). To this end, new variables (slack variables) are introduced to rewrite the inequality.

$$\sum_{j=1}^{n} c_j x_j \Rightarrow z = v + \sum_{j=1}^{n} c_j x_j$$
$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \Rightarrow x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_i$$
$$x_j \geq 0 \text{ for } j = 1, 2, \ldots, n+m$$

Later, we have something called the dual of a linear program, which is used to show that the SIMPLEX algorithm actually computes the optimal solution to the problem.

$$\begin{aligned} &\text{minimize} \\ &\qquad \sum_{i=1}^{m} b_i y_i \\ &\text{subject to} \\ &\qquad \sum_{i=1}^{m} a_{ij} y_i \geq c_j \text{ for } j = 1, 2, \ldots, n \\ &\qquad y_i \geq 0 \text{ for } i = 1, 2, \ldots, m. \end{aligned}$$

2.4.2 Randomized Algorithms

An algorithm that uses random numbers to decide what to do next anywhere in its logic is called a Randomized Algorithm. Typically, this randomness is used to reduce time complexity or space complexity in other standard algorithms.

- RandQS: in Randomized Quick Sort, we use a random number to pick the next pivot.

  P_{ij} is the probability of the event and we are interested in the expected number of comparisons:

$$E[\#\text{comparisons}] = E\left[\sum_{i<j} X_{ij}\right] = \sum_{i<j} E\left[X_{ij}\right] = \sum_{i<j} p_{ij}$$
$$E\left[X_{ij}\right] = (1 - p_{ij}) \cdot 0 + p_{ij} \cdot 1 = p_{ij}$$

a	$\cdots$	i	$\cdots$	j	$\cdots$	b

Suppose that we divide a sorted list is into three segments a-i, i-j, and j-b. besides,a-i and j-b do not affect p_{ij}.The p_{ij} is calculated by the probability of comparing i with j in the i-j segment. We substitute $p_{ij} = \frac{2}{j+1-i}$ and then:

$$\begin{aligned} E[\#\text{comparisons}] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &= 2 \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{1}{k} \\ &\leq 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{1}{k} \\ &= 2 \sum_{i=1}^{n-1} H_n \\ &< 2nH_n = O(n \log n) \end{aligned}$$

- Randomized Min-cut:

  The algorithms Contract selects uniformly at random one of the remaining edges and contracts this edge until two vertices remain. The cut determined by this algorithm contains precisely the edges that have not been contracted. Counting the edges between the remaining two vertices yields an estimate of the size of the minimum cut of G

  And for any min-cut C, the probability that RandMinCut(G) returns C is $\geq \frac{2}{n(n-1)}$ calling RandMinCut $t\frac{n(n-1)}{2}$ times

References

[1] Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein, Introduction to Algorithms, third edition