Final Exercise

TA Time

A – Markov Matrix

$$\begin{bmatrix} 90\% & 20\% \\ 10\% & 80\% \end{bmatrix}\begin{bmatrix} 3/4 \\ 1/4 \end{bmatrix}=\begin{bmatrix} 90\%\times 3/4+20\%\times 1/4 \\ 10\%\times 3/4+80\%\times 1/4 \end{bmatrix}=\begin{bmatrix} 29/40 \\ 11/40 \end{bmatrix}$$

- *Thought*
  - Keep multiplying transition matrix until:
    - The first city's value is equal or below p
    - The first city's value is **equal or larger than the original value after the operation**

```
int transition(double x[10])
{

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            x1[i] += transition_matrix[i][j] * x[j];
        }
    }

    if(x1[0] >= x[0]) return 0;
    else if(x1[0] <= p) return 1;
    else return transition(x1);
}
```

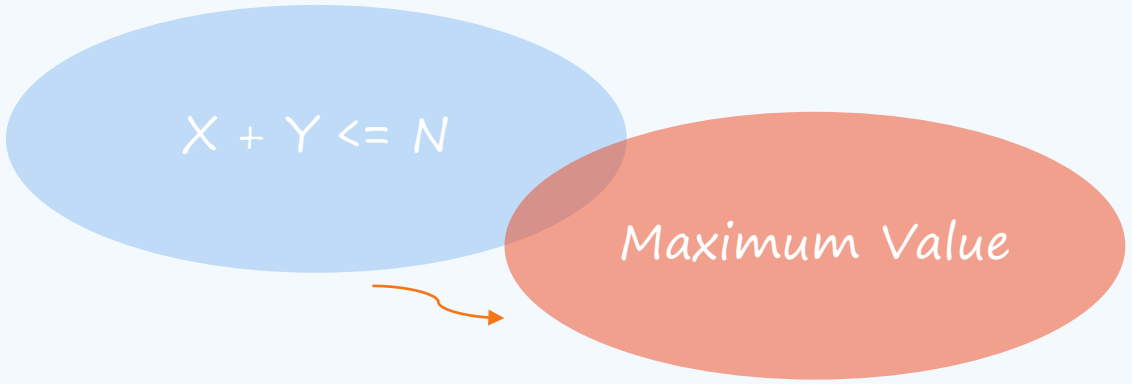
B - Great Depression Again

Sample Input :

10 3 4
TOYOTA 10 100
GM 20 90
FORD 30 80
Volkswagen 40 70
Daimler 50 60
Honda 60 50
Nissan 70 40
PEUGEOT 80 30
FIAT 90 20
BMW 100 10

Sample Output :

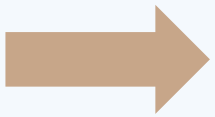
BMW
FIAT
PEUGEOT



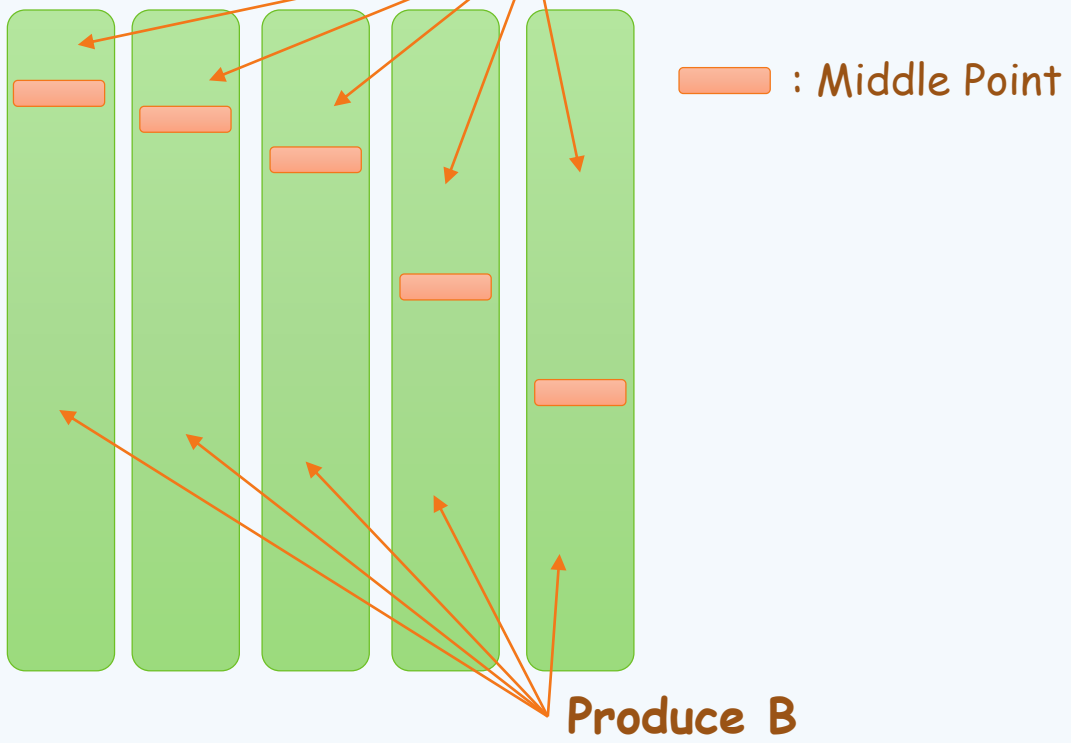
Instinctively,

BMW	200 100
BENZ	100 100
Toyota	100 200

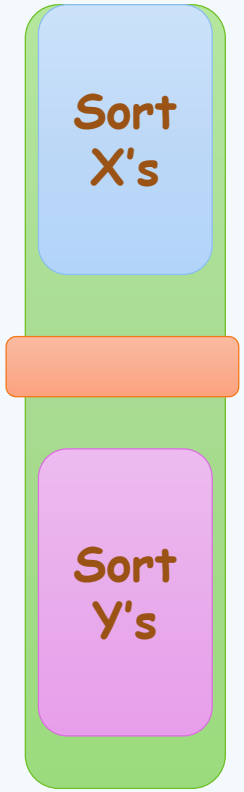
A > B , Producing only A



Sort by A-B



However, this time we have limited resource. Which means (X + Y) may smaller than N.
(Produce only X type-A and Y type-B)



We sort the company that best produce type-A car by the value of A

Same with type-B by value B

So far we split the whole list at middle, and get a total value of (X's+Y's).
The maximum total value is the best combination.

```
struct Fac {
    char s[24];
    int a, b;
} fac[N];
```

```
sort(fac, fac+n, cmp0);

int ans=-1, mid=-1;
for(int i=0; i < n; i ++) {
    int tmp = split(i);
    if(tmp > ans) ans=tmp, mid=i;
}

split(mid);
sort(lst_a, lst_a+x, cmp3);
```

```
int cmp0(Fac a, Fac b) {
    return a.a-a.b > b.a-b.b;
}
int cmp1(int a, int b) {
    return fac[a].a > fac[b].a;
}
int cmp2(int a, int b) {
    return fac[a].b > fac[b].b;
}
int cmp3(int a, int b) {
    return strcmp(fac[a].s, fac[b].s)<=0;
}
```

```
split(int mid){

    for(int i = 0; i < mid; i ++) list_a[lenth_a ++] = i;
    for(int j = mid; j < n; j ++) list_b[lenth_b ++] = j;

    sort(lst_a, lst_a+len_a, cmp1);
    sort(lst_b, lst_b+len_b, cmp2);

    for(int i = 0; i < x; i ++) tmp += fac[list_a[i]].a;
    for(int i = 0; i < y; i ++) tmp += fac[list_b[i]].b;
    return tmp;
}
```

DO NOT DISAPPOINT HIM!!

C - Stable Sort

- **Definition**
  - When you meet the same value, you will not change their original location

- **Method**
  - You can use **qsort** and remember the enter of the order

```
int compare(const void* a, const void* b)
{
    const Data* qa = (const Data *)a;
    const Data* qb = (const Data *)b;
    if(qa->value - qb->value != 0) return qb->value - qa->value;
    else return qa->order - qb->order;
}
```

D - Determinant

$$\begin{vmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ a & b & c & d \end{vmatrix} = a\begin{vmatrix} f & g & h \\ j & k & l \\ b & c & d \end{vmatrix} - b\begin{vmatrix} e & g & h \\ i & k & l \\ a & c & d \end{vmatrix} + c\begin{vmatrix} e & f & h \\ i & j & l \\ a & b & d \end{vmatrix} - d\begin{vmatrix} e & f & g \\ i & j & k \\ a & b & c \end{vmatrix}$$

$$|A| = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

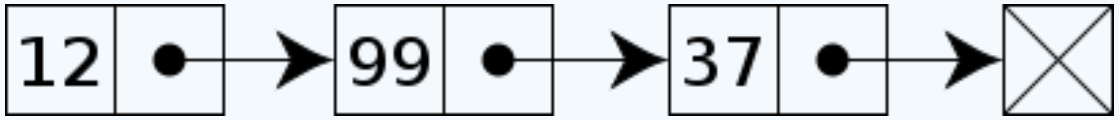
$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc.$$

Termination

```
long long int det(int N, int* matrix)
{
    if(N == 2){
        return matrix[0]*matrix[3] - matrix[1]*matrix[2];
    }
    else{
        int i, j, k;
        long long int ans = 0;
        int tmp[N-1][N-1];
        for(i = 0; i < N; i++){
            for(j = 0; j < i; j++)
                for(k = 1; k < N; k++)
                    tmp[k-1][j] = matrix[k][j];
            for(j = i; j < N-1; j++)
                for(k = 1; k < N; k++)
                    tmp[k-1][j] = matrix[k][j+1];
            if(i%2 == 0)    ans += matrix[i] * det(n-1,b);
            else    ans -= matrix[i] * det(n-1,b);
        }
        return ans;
    }
}
```

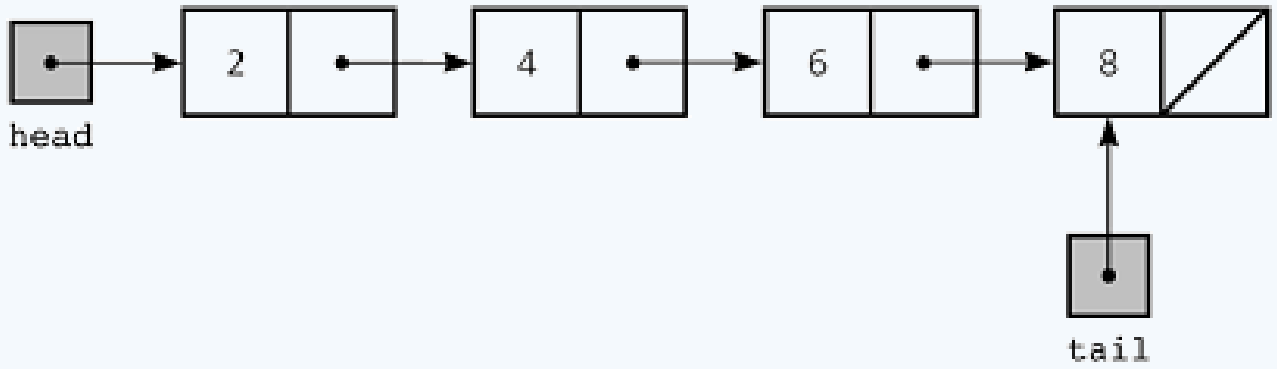
E - Reverse Linked List

```c
typedef struct Single {
    int data;
    struct Node *next;
} Node;
```




```c
typedef struct Double {
    int data;
    struct Node *prev;
    struct Node *next;
} Node;
```

```
Node *head = NULL;
```




```
Node *tail = NULL;
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "function.h"

int main() {
    Node *head = NULL;
    int n, data, i;

    scanf("%d", &n);
    for( i=0; i<n; i++ ) {
        scanf("%d", &data);
        head = Create_List( head, data );
    }

    head = Reverse_List( head );

    Print_List( head );
    Free_List( head );

    return 0;
}
```

Create List

```c
Node* Create_List(Node *head, int data){
    Node *p = (Node*)malloc(sizeof(Node));
    p->data = data;
    p->next = NULL;
    if(head == NULL)    head=p;
    else{
        Node *tmp = head;
        while(tmp->next != NULL){
            tmp = tmp->next;
        }
        tmp->next = p;
    }
    return head;
}
```

```c
Node *Create_List(Node* head, int data)
{
    Node *p = (Node*)malloc(sizeof(Node));
    p->data = data;
    p->next = NULL;
    if(head == NULL){
        p->pre = NULL;
        head = p;
    }
    else{
        Node *tmp = head;
        while(tmp->next != NULL){
            tmp = tmp->next;
        }
        tmp->next = p;
        (tmp->next)->pre = tmp;
    }
    return head;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "function.h"

int main() {
    Node *head = NULL;
    int n, data, i;

    scanf("%d", &n);
    for( i=0; i<n; i++ ) {
        scanf("%d", &data);
        head = Create_List( head, data );
    }

    head = Reverse_List( head );

    Print_List( head );
    Free_List( head );

    return 0;
}
```

Reverse List

```
Node* Reverse_List(Node *head){
    Node *prev = NULL;
    Node *tmp;
    Node *curr = head;
    while(curr){
        tmp = curr->next;
        curr->next = prev;
        prev = curr;
        curr = tmp;
    }
    return prev;
}
```

```
Node* Reverse_List(Node* head)
{
    Node *tmp = head;
    while(tmp->next!=NULL){
        tmp = tmp->next;
    }
    return tmp;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "function.h"

int main() {
    Node *head = NULL;
    int n, data, i;

    scanf("%d", &n);
    for( i=0; i<n; i++ ) {
        scanf("%d", &data);
        head = Create_List( head, data );
    }

    head = Reverse_List( head );

    Print_List( head );
    Free_List( head );

    return 0;
}
```

Print List

```c
void Print_List(Node *head){
    while(head->next!=NULL){
        printf("%d->",head->data);
        head = head->next;
    }
    printf("%d\n",head->data);
}
```

```c
void Print_List(Node* tail)
{
    while(tail->pre!=NULL)
    {
        printf("%d->",tail->data);
        tail = tail->pre;
    }
    printf("%d\n", tail->data);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "function.h"

int main() {
    Node *head = NULL;
    int n, data, i;

    scanf("%d", &n);
    for( i=0; i<n; i++ ) {
        scanf("%d", &data);
        head = Create_List( head, data );
    }

    head = Reverse_List( head );

    Print_List( head );
    Free_List( head );

    return 0;
}
```

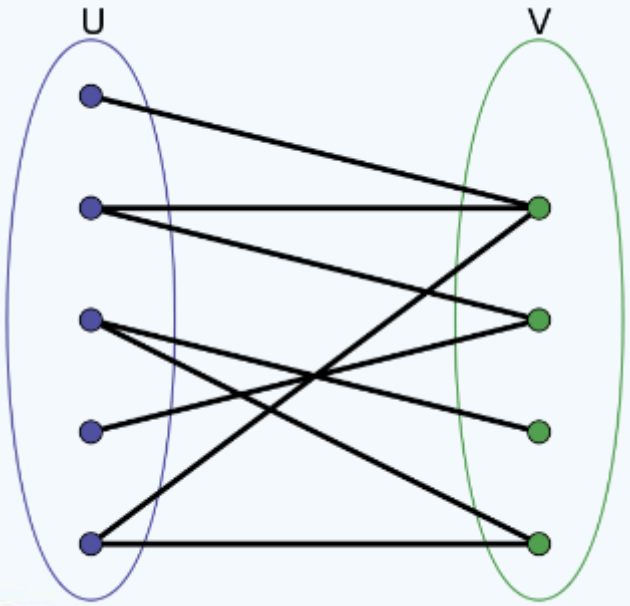
Free List

```c
void Free_List(Node* head)
{
    while (head != NULL)
    {
        Node* tmp = head;
        head = head->next;
        free(tmp);
    }
}
```

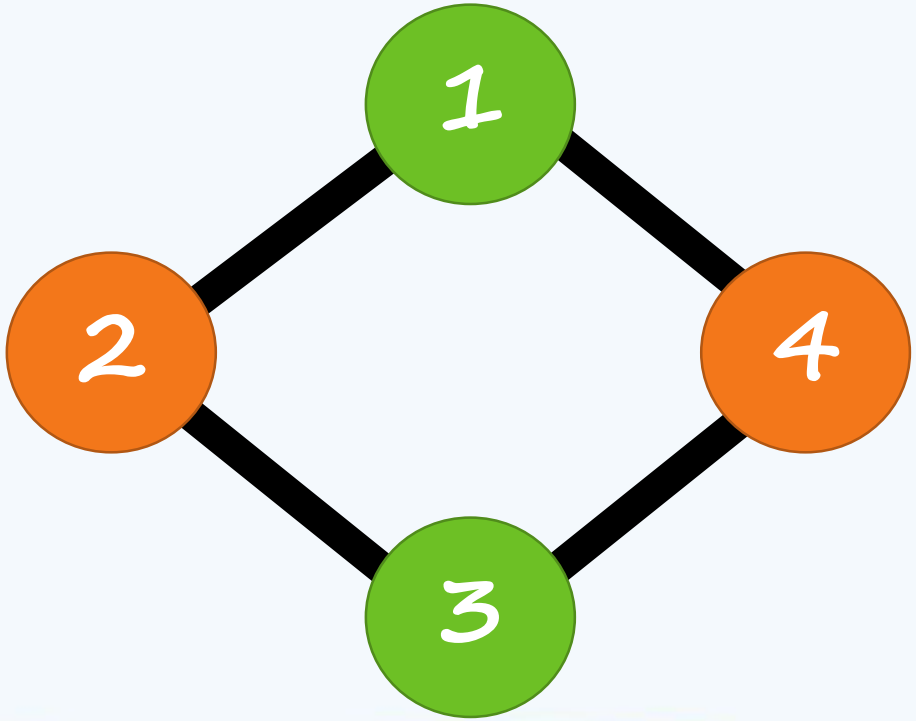
```c
void Free_List(Node* tail)
{
    while (tail != NULL)
    {
        Node* tmp = tail;
        tail = tail->prev;
        free(tmp);
    }
}
```

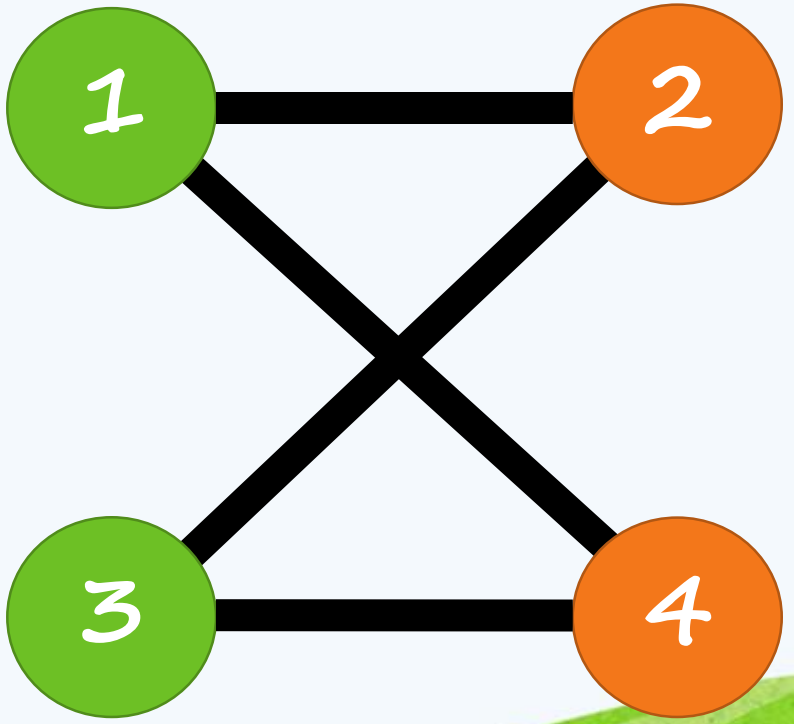
F - Bipartite Graph

- **Definition**
  - All vertices can be divide into two disjoint and independent sets U and V
  - such that every edge connects a vertex in U to one V



- Method – Coloring
  - Any two adjacent vertices need to have different color





```
while(m--){
    scanf("%d %d" ,&vertice_1 , &vertice_2);
    a[vertice_1][vertice_2] = 1 ;
    a[vertice_2][vertice_1] = 1 ;
}

if(color[vertice] == -1) color[vertice] = mark;
else if(color[vertice]!=mark)return 0 ;
else return 1 ;

for(i = 1 ; i <= n ; i++ ){
    if(a[vertice][i] == 1){
        ans &=  coloring(i , ~mark) ;
    }
}
```

G - XOR

- XOR
  - Exclusive or
- Example
  - 0101 xor 0011 = 0110 (Decimal 6)
  - 0010 xor 1010 = 1000 (Decimal 8)

A	B	⊕
F	F	F
F	T	T
T	F	T
T	T	F

True table

- SUM = (a1 ^ k) + (a2 ^ k) + … + (an ^ k)
  - SUM is minimum

- Sample Input : 1 2 3 4 5 6
  - 001^k + 010^k + 011^k + 100^k + 101^k + 110^k = SUM
  - Sample Output : 21

- Another example: 0 1 1 1 1
  - 0^1 + 1^1 + 1^1+ 1^1+ 1^1
  - Sample Output : 1

```
for (i = 0; i < Maxdigits; i ++) {
    zeros, ones = 0;
    for(j = 0; j < n; j ++) {
        if ( bit[j][i] == 0 ) zeros ++;
        else ones ++;
    }
    if(zeros > ones) ans += ones*base2;
    else ans += zeros*base2;
    base2 *= 2;
}
```

Thanks~