1) Stack and Queue

- Понимать и уметь применять базовые операции pop(), push()

- На всякий случай знать как имплементировать через linked list

Stack

```
void Push(int data){
      Node *node = new Node(data);
      if (Empty()){} else {
         node -> next = top;
      }
         top = node;
         size++;
   }
   void Pop(){
      if (!Empty()){
         top = top->next;
         size--;
      }
   }
```

Queue

```
void Push(int data) {
      Node *node = new Node (data);
      if (Empty()){
         head = node;
      } else {
         tail->next = node;
      }
         tail = node;
         size++;
   }
   void Pop() {
      if (!Empty()){
         head = head ->next;
         size--;
      }
   }
```

2) Linked List

- добавление/удаление спереди,

```
void PushFront(int data){
      Node *node  = new Node(data);
      if (Empty()){head = tail = node;} else {
         head->prev = node;
         node->next = head;
```

```
        head = node;
      }
      size++;
   }
void PopFront() {
      if (!Empty()){
         if (Size()==1){
            size = 0;
            head = tail = NULL;
         } else {
            head->next->prev = NULL;
            head = head->next;
            size--;
         }
      }
   }
```

- добавление/удаление в конец,

```
void PushBack(int data){
      Node *node = new Node(data);
      if (Empty()){head = tail = node;} else {
         node->prev = tail;
         tail->next = node;
         tail = node;
      }
      size++;
   }
void PopBack() {
      if (!Empty()) {
         if (Size() == 1){
            size = 0;
            head = tail = NULL;
         } else {
            tail->prev->next = NULL;
            tail = tail->prev;
            size--;
         }
      }
   }
```

- reverse,

```
void ReverseLinkedList(){
      LinkedList *list = new LinkedList();
      Node *node = new Node(0);
      node = head;
      for (int i=0;i<Size();i++){
         list->PushFront(node->data);
         node = node->next;
```

```
    }
    head=list->head;
    tail=list->tail;
  }
```

- print,

```
void Print() {
    Node *node = new Node(0);
    node = head;
    while (node->next!=NULL){
      cout<<node->data<<"->";
      node = node->next;
    }
    cout<<node->data<<endl;
  }
```

- delete nth element;

```
void Delete(int data){
    Node *node = new Node(data);
    node = head;
    if (data>Size()||data==0){
      cout<<"No such element";
      return;
    };
    if (data==1)PopFront();
     else
    if (data==Size())PopBack(); else
    {
    for (int i=1;i<data;i++){
      node = node->next;
    }
    node->prev->next = node->next;
    node->next->prev = node->prev;
    node->next = NULL;
    node->prev = NULL;
    size--;
    }
    Print();
  }
```

- search nth element;

```
void search(int data){
    Node *node = new Node(data);
    node = head;
    if (data>Size()||data==0){
      cout<<"No such element";
```

```
        return;
    };
    if (data==1)cout<<head->data;
     else
    if (data==Size())cout<<tail->data; else
    {
    for (int i=1;i<data;i++){
        node = node->next;
    }
    cout<<node->data;
}
```

- знать за какое время производятся эти операции.

- Разница между Linked List и Doubly Linked List

Single linked list когда ты имеешь ссылку только на след элемент, double LL хранишь ссылку и на след и пред элемент

3) Prime numbers

- Факторизация - разложение числа на prime numbers (могут попасться какие-нибудь математические вопросы или задачи, простые)

```
vector<int> factorize(int x) {
    vector<int> factors;

    for (int i = 2; i <= sqrt(x); i++) {
        while (x % i == 0) {
            factors.push_back(i);
            x /= i;
        }
    }

    if (x != 1) {
        factors.push_back(x);
    }

    return factors;
}
```

- Решето Эратосфена O (n*log(log(n)))

```
int p [1000000];

void sieve(int n){
    for (int i=2; i*i<=n;i++){
```

```
        if (p[i]==0){
            int k=i;
            while (k+i<=n){
                k+=i;
                p[k]=1;
            }
        }
    }
    for (int i=2;i<=n;i++){
        if (p[i]==0)
        cout<<i<<" ";
    }
}
```

4) Quicksort

```
int a[1000];
void QuickSort(int l, int r){
    int i=l;
    int j=r;
    int p = a[(l+r)/2];
    while (i<j){
        while (a[i]<p)i++;
        while (a[j]>p)j--;
        if (i<=j){
            swap(a[i], a[j]);
            i++;
            j--;
        }
    }
    if (l<j) QuickSort(l,j);
    if (i<r) QuickSort(i,r);
}
```

- какие операции производятся при сортировке массива по partition element

- понимание рекурсии например, сколько раз вызывается рекурсия на каком-то примере, и как сортируется массив

- худший случай если в качестве опорного на каждом этапе будет выбран элемент либо наименьший, либо наибольший из всех обрабатываемых n*n,

- лучший случай n*log2(n) время работы

5) Mergesort

```
void Merge(long int l, long int m, long int r){
    long int uk1=0, uk2=0, k=l;
    long int left = m-l+1;
    long int right = r-m;
```

```
    long int L1[left];
    long int R1[right];

    for (long int i=0;i<left;i++)
        L1[i] = a[i+l];
    for (long int i=0;i<right;i++)
        R1[i] = a[m+i+1];
    while (uk1<left && uk2<right){
        if (L1[uk1]<R1[uk2]){
            a[k] = L1[uk1];
            uk1++;
        } else
        {
            a[k]=R1[uk2];
            uk2++;
        }
        k++;
    }
    while (uk1<left){
        a[k] = L1[uk1];
        uk1++;
        k++;
    }
    while (uk2<right){
        a[k] = R1[uk2];
        uk2++;
        k++;
    }
}

void MergeSort(long int l, long int r){
    if (l<r){
        long int m = (l+r)/2;
        MergeSort(l,m);
        MergeSort(m+1,r);
        Merge(l,m,r);
    }
}
```

- понимание рекурсии и как сортируется массив, операции merge()
- лучшее n*log2(n) и худшее время работы

6) Binary Search

```
int p[1000000];

int main(){
    int n,k,left,right,mid;
```

```
   cin>>n;
   for (int i=0;i<n;i++)cin>>p[i];
   cin>>k;
   left=0;
   right=n-1;
   while(left<right){
      mid=(left+right)/2;
      if (p[mid]<k)left=mid+1; else
      right=mid;
   }
   //вывести последний экземпляр
   if (p[right]==k){
      while(p[right+1]==k)right++;
      cout<<++right;
   } else
   //вывести первый экземпляр
   //if (p[right]==k)cout<<right+1; else
   {
   cout<<"not found"<<endl<<"Could be on "<<left+1<<" place"<<endl<<"Closest i
s ";
   int a=p[right-1];
   int b=p[right];
   if (k-a==b-k)cout<<a<<" and "<<b; else
   if (k-a<b-k)cout<<a; else cout<<b;
   }
}
```

- знание и понимание времени работы алгоритма

- the leftmost (самое первое вхождение элемента),

- rightmost (самое последнее вхождение элемента)

- average case (вхождение элемента в любом промежутке от самого первого до самого последнего вхождения) binary search

7) BST

- добавление/удаление, поиск элемента

```
   Node *insert_node(Node *node, int data){
      if (node == NULL)node = new Node(data); else
      if (node->data>data)
         node->left = insert_node(node->left,data);
         else
         node->right = insert_node(node->right,data);
   }

Node *deleteNumber(Node *node, int data){
      if (node==NULL)return NULL;
```

```
    if (node->data>data)
        node->left = deleteNumber(node->left, data); else
    if (node->data<data)
        node->right = deleteNumber(node->right, data);
    else {
        if (node->right==NULL && node->left==NULL)
            node = NULL; else
        if (node->left == NULL)
            node = node->right; else
        if (node->right == NULL)
            node = node->left; else {
                Node *t = findMin(node->right);
                node->data = t->data;
                node->right = deleteNumber(node->right,t->data);
            }
    }
    return node;
}
```

- понимание термина “сбалансированное дерево” - высота дерева зависит от порядка добавления элементов

- print in order, print post order, на всякий случай print level-wise (bfs)

```
void inOrder(Node *node){
    if (node==NULL)return;
    inOrder(node->left);
    cout<<node->data<<" ";
    inOrder(node->right);
}
```

8) Heap + Heap Sort

- insertion to heap (minHeap and maxHeap respectively),

```
void insert (int k){
    sz++;
    a.push_back(k);
    int i = sz-1;
    while (i>0 && a[parent(i)]>a[i]){
        swap(a[parent(i)], a[i]);
        i = parent(i);
    }
}
void insert(int i){
    size++;
    a.push_back(i);
    int k = size-1;
```

```
    while (k>0 && a[Parent(k)]<a[k]){
        swap(a[Parent(k)], a[k]);
        k = Parent(k);
    }
}
```

- extract maximum from heap, heapify

```
void heapify(int i){
    if (Left(i)>size - 1)return;
    int j = Left(i);
    if (a[j]<a[Right(i)])j = Right(i);
    if (a[i]<a[j]){
        swap(a[i],a[j]);
        heapify(j);
    }
}
void extractMax(){
        swap(a[0],a[size-1]);
        size--;
        heapify(0);
}
```

-extract minimum from heap, heapify

```
void heapify(int i){

    if (left(i)>sz-1)return;
    int j = left(i);
    if (a[j]>a[right(i)] && right(i)<sz)
    j = right(i);
    if (a[i]>a[j]){
        swap(a[i], a[j]);
        heapify(j);
    }
}

void extractMin(){
    swap(a[0], a[sz-1]);
    sz--;
    heapify(0);
}
```

- знание времени добавления в heap и удаления из heap

O(1) best, O(log n) average and worst

- Heap Sort (студенты Акшабаева не проходили)

9) Rabin-Karp

- Умение вычислять хэш строки и подстроки, знание времени работы алгоритма

```
int get_hash(string s){
    int hash = 0;
    int p = 31;
    int p_pow = 1;
    for (int i=0;i<s.size();i++){
        hash += s[i]*p_pow;
        p_pow *= p;
    }
    return hash;
}

vector <int> get_hash(string s){
    int n = s.size();
    vector <int> hash(n);
    hash[0]=s[0];
    int p = 31;
    int p_pow = 1;
    for (int i=1;i<n;i++){
        p_pow *= p;
        hash[i] += hash[i-1] + s[i]*p_pow;
    }
    return hash;
}

int main(){
    string s,t;
    cin>>s>>t;
    vector <int> hash_s = get_hash(s);
    int hash_t = get_hash(t)[t.size()-1];
    int n = s.size();
    vector <int> p(n);
    p[0] = 1;
    for (int i=1;i<n;i++)
        p[i] = p[i-1] * 31;
    for (int i=0; i<s.size()-t.size()+1;i++){
        int j = i+t.size()-1;
        int hash_i_j = hash_s[j];
        if (i>0)
            hash_i_j = hash_s[j] - hash_s[i-1];
        if (hash_i_j == hash_t * p[i])
            cout<<i<<" ";
    }
    return 0;
}
```

10) KMP

- умение вычислять префикс функцию

```
vector <int> prefix_func (string s){
    int n = s.size();
    vector <int> p(n);
    p[0] = 0;
    for (int i=1;i<n;i++){
        int j=p[i-1];
        while (j>0 && s[i]!=s[j])j=p[j-1];
        if (s[i]==s[j])j++;
        p[i]=j;
    }
    return p;
}

int main(){
    string s, t;
    cin >> s >> t;
    string s1 = t + "#" + s;
    vector<int> p = prefix_func(s1);
    for (int i = 0; i < s1.size(); i++){
        if (p[i] == t.size())
            cout << i - t.size() - t.size() << " ";
    }
    return 0;
}
```

11) Trie

- Insertion/Deletion, Search

```
void insert (char a, chas s){
        Node *cur = root;
            Node *node = new Node(s);
            cur->ch[a-'a']->ch[s-'a'] = node;
                cur = node;
                cnt++;
    }
```

12) Adjacency matrix / Edge List

- понимать как представлять граф с помощью списка ребер и матрицы смежности

```
int g[100][100];
int n,m,x,y;

int main(){
    cin>>n>>m;
    for (int i=0;i<m;i++){
        cin>>x>>y;
        x--; y--;
        g[x][y] = 1;
        g[y][x] = 1;
    }
    for (int i=0;i<n;i++){
        for (int j=0;j<n;j++)
            cout<<g[i][j]<<" ";
        cout<<endl;
    }
    return 0;
}
```

```
vector <int> g[100];
int n,m,x,y;

int main(){
    cin>>n>>m;
    for (int i=0;i<m;i++){
        cin>>x>>y;
        x--;y--;
        g[x].push_back(y);
        g[y].push_back(x);
    }

    for (int i=0;i<n;i++){
        cout<<i+1<<"---";
        for (int j=0;j<g[i].size();j++)
            cout<<g[i][j]+1<<" ";
        cout<<endl;
    }
    return 0;
}
```

13) BFS

- Output of BFS traversal - понимание

```
vector <int> g[100];
int n,m,x,y;
int used[100];
int d[100];
queue <int> q;

void bfs(int v){
   q.push(v);
   used[v]=1;
   d[v]=0;
   while (!q.empty()){
      x = q.front();
      for (int i=0;i<g[x].size();i++){
         y = g[x][i];
         if (used[y]==0){
            q.push(y);
            used[y]=1;
            d[y]=d[x]+1;
         }
      }
      q.pop();
   }

}

int main(){
   cin>>n>>m;
   for (int i=0;i<m;i++){
      cin>>x>>y;
      x--; y--;
      g[x].push_back(y);
      g[y].push_back(x);
   }
   bfs(0);
   for (int i=0;i<m;i++)
      cout<<i<<"---"<<d[i]<<endl;
   return 0;
}
```

- понимание логики и имплементации алгоритма

- На всякий случай знать время работы

$O(n+m)$, где n— число вершин, m— число рёбер

14) DFS

- Output of DFS traversal - понимание

```
int used[100];
vector <int> g[100];
int n,m,x,y;

void dfs(int v){
   used[v]=1;
   for (int i=0;i<g[v].size();i++){
      y = g[v][i];
      if (used[y]==0)dfs(y);
   }

}

int main(){
   cin>>n>>m;
   for (int i=0;i<m;i++){
      cin>>x>>y;
      x--; y--;
      g[x].push_back(y);
      g[y].push_back(x);
   }
   int cnt=0;
   for (int i=0; i<n; i++)
      if (used[i]==0){
         dfs(i);
         cnt++;
      }
   cout<<cnt;
   return 0;
}
```

- понимание логики и имплементации алгоритма

- На всякий случай знать время работы

$O(n+m)$, где n— число вершин, m— число рёбер

15) Topological Sort

- Вывод топологической сортировки может быть разный.

```
topological_sort(N, adj[N][N])
      T = []
      visited = []
      in_degree = []
      for i = 0 to N
```

```
            in_degree[i] = visited[i] = 0

      for i = 0 to N
            for j = 0 to N
                  if adj[i][j] is TRUE
                        in_degree[j] = in_degree[j] + 1

      for i = 0 to N
            if in_degree[i] is 0
                  enqueue(Queue, i)
                  visited[i] = TRUE

      while Queue is not Empty
            vertex = get_front(Queue)
            dequeue(Queue)
            T.append(vertex)
            for j = 0 to N
                  if adj[vertex][j] is TRUE and visited[j] is FALSE
                        in_degree[j] = in_degree[j] - 1
                        if in_degree[j] is 0
                              enqueue(Queue, j)
                              visited[j] = TRUE
      return T;
}

T = []
visited = []

topological_sort( cur_vert, N, adj[][] ){
   visited[cur_vert] = true
   for i = 0 to N
      if adj[cur_vert][i] is true and visited[i] is false
      topological_sort(i)
   T.insert_in_beginning(cur_vert)
}
```

16) Cycles in graph O (m)

- понимание термина “цикл”, уметь находить и считать циклы в графе

- понимать как находить цикл в графе программно:

```
int n;
vector < vector<int> > g;
vector<char> cl;
vector<int> p;
int cycle_st, cycle_end;

bool dfs (int v) {
        cl[v] = 1;
```

```
	for (size_t i=0; i<g[v].size(); ++i) {
		int to = g[v][i];
		if (cl[to] == 0) {
			p[to] = v;
			if (dfs (to))  return true;
		}
		else if (cl[to] == 1) {
			cycle_end = v;
			cycle_st = to;
			return true;
		}
	}
	cl[v] = 2;
	return false;
}

int main() {
	... чтение графа ...

	p.assign (n, -1);
	cl.assign (n, 0);
	cycle_st = -1;
	for (int i=0; i<n; ++i)
		if (dfs (i))
			break;

	if (cycle_st == -1)
		puts ("Acyclic");
	else {
		puts ("Cyclic");
		vector<int> cycle;
		cycle.push_back (cycle_st);
		for (int v=cycle_end; v!=cycle_st; v=p[v])
			cycle.push_back (v);
		cycle.push_back (cycle_st);
		reverse (cycle.begin(), cycle.end());
		for (size_t i=0; i<cycle.size(); ++i)
			printf ("%d ", cycle[i]+1);
	}
}
```

17) Prima

- понимание понятий spanning tree, minimal spanning tree

```
#define pb push_back
#define mp make_pair

using namespace std;
vector<pair<int, int> > g[100];
bool used[100];
```

```
int d[100];
int c[100];
int n, m, x, y, l;

int main() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        cin >> x >> y >> l;
        x--;
        y--;
        g[x].pb(mp(y, l));
        g[y].pb(mp(x, l));
    }

    for (int i = 0; i < n; i++) {
        d[i] = -1;
        d[0] = 0;
    }

    for (int i = 0; i < n; i++) {
        int v = -1;
        for (int j = 0; j < n; j++)
            if (used[j] == false)
                if (v == -1 || d[v] > d[j] || d[j]!=-1)
                    v = j;
        used[v] = true;
        for (int j = 0; j < g[v].size(); j++) {
            int u = g[v][j].first;
            l = g[v][j].second;
            if (used[u]==false && (d[u] == -1 || d[u] > l)) {
                d[u] = l;
                c[u] = v;
            }
        }
    }
    for (int i = 0; i < n; i++)
        if (c[i] != i)
            cout << i + 1 << " " << c[i] + 1 << endl;

    return 0;
}
```