Contents

1 Math

1.1 Power

```
#define MOD 100000007

int power(int a, int n) {
  int ret = 1;

  while(n) {
    if (n % 2) {
      ret = (ret * a) % MOD;
    }

    a = (a * a) % MOD;
    n /= 2;
  }

  return ret;
}
```

1.2 Extended Euclidean Algorithm

```
tuple<int, int, int> eea (int a, int b) {
  int s1 = 1, s2 = 0, t1 = 0, t2 = 1, r1 = a, r2 = b, q;
  while (r1 > 0) {
    q = r1 / r2;
    tie(r1, r2) = make_tuple(r2, r1 - q * r2);
    tie(s1, s2) = make_tuple(s2, s1 - q * s2);
    tie(t1, t2) = make_tuple(t2, t1 - q * t2);
  }

  return make_tuple(r1, s1, t1);
}
```

1.3 Euler's Phi Function

```
int euler_phi (int x) {
  int ret = x;
  for (int i = 2; i * i <= x; i++) {
    if (x % i == 0) {
      while (x % i == 0)
        x /= i;

      ret -= ret / i;
    }
  }

  if (x > 1)
    ret -= ret / x;

  return ret;
}
```

2 String

2.1 KMP

```
void get_fail(const char* p, int len, vector<int>& fail) {
    fail.resize(len);
    fail[0] = 0;
    for (int i = 1, j = 0; i < len; ++i) {
        while (j && p[i] != p[j]) j = fail[j - 1];
        if (p[i] == p[j]) fail[i] = ++j;
    }
}

void KMP(const char* text, int tlen, const char* pattern, int plen, vector<int>& fail, vector<int>& ans) {
    ans.clear();
    for (int i = 0, j = 0; i < tlen; ++i) {
        while (j && text[i] != pattern[j]) j = fail[j - 1];
        if (text[i] == pattern[j]) {
            if (j == plen - 1) {
                ans.push_back(i - j);
                j = fail[j];
            } else
                ++j;
        }
    }
}
```

3 Data Structure

3.1 Segment Tree

```
long long init(int index, int start, int end){
    if (start == end)
        tree[index] = A[start];
    else{
        int mid = (start+end)/2;
        tree[index] = init(index*2+1, start, mid) + init(index*2+2, mid+1, end);
    }
    return tree[index];
}

long long sum(int index, int start, int end, int left, int right){
    // 구간이전혀겹치지않는경우
    if (start > right || end < left)
        return 0;
    else if (left <= start && end <=right)
        return tree[index];
    else {
        int mid = (start+end) / 2;
        return sum(index*2+1, start, mid, left, right) + sum(index*2+2, mid+1, end, left, right);
    }
}

void update(int changed_index, long long diff, int index, int start, int end){
    if (changed_index < start || changed_index > end)
        return;
    tree[index] += diff;

    if (start != end){
        int mid = (start+end) / 2;
        update(changed_index, diff, index*2+1, start, mid);
        update(changed_index, diff, index*2+2, mid+1, end);
    }
}
```

4 Graph

4.1 DFS and BFS

```
#define MAX 100005

bool visited[MAX];
vector<int> g[MAX];
queue<int> q;

void dfs(int start) {
  visited[start] = true;
```

```
  for (auto& i : g[start]) {
    if (!visited[i]) {
      visited[i] = true;
      dfs(start);
    }
  }
}

void bfs(int start) {
  queue.push(start);
  visited[start];

  while(!q.empty()) {
    int now = q.front();
    q.pop();

    for (auto& i : g[now]) {
      if (!visited[i]) {
        visited[i] = true;
        q.push(i);
      }
    }
  }
}
```

4.2 0-1 BFS

```
#define MAX 100005

deque<int> dq;
vector<int> g[MAX];
bool visited[MAX] = { 0 };

void bfs(int start) {
  dq.push_back(start);
  visited[start] = true;

  while(!dq.empty()) {
    int dq_size = dq.size();

    while(dq_size) {
      int item = dq.front();
      dq.pop_front();

      for (auto& w : g[item]) {
        if ((w == zero_value) && !visited[w]) {
          visited[w] = true;
          dq.push_front(w);
        }

        else {
          visited[w] = true;
          dq.push_back(w);
        }
      }

      dq_size--;
    }
  }
}
```

4.3 Bipartite Graph

```
#define RED 1
#define BLUE 2
#define MAX 100005

vector<int> g[20005];
queue<int> q;
int visited[20005] = { 0 };

void bfs(int start) {
  int color = RED;

  visited[start] = RED;
  q.push(start);

  while(!q.empty()) {
    int now = q.front();
```

```
    q.pop();

    if (visited[now] == RED)
      color = BLUE;

    else
      color = RED;

    for (auto& i : g[start]) {
      if (!visited[i]) {
        visited[i] = color;
        q.push(i);
      }
    }
  }
}

bool check_bipartite() {
  for (int i = 1; i <= vertex_num; i++) {
    for (auto& j : g[i]) {
      if (visited[i] == visited[j])
        return false;
    }
  }

  return true;
}
```

4.4 Union Find

```
int fi(int a) {
  if (a = parent[a])
    return a;

  else
    return parent[a] = find(parent[a]);
}

void uni(int a, int b) {
  a = fi(a);
  b = fi(b);

  if (rank[a] == rank[b]) {
    rank[a]++;
    parent[b] = a;
    return;
  }

  if (rank[a] > rank[b]) {
    parent[b] = a;
    return;
  }

  parent[a] = b;
}
```

4.5 Dijkstra's Algorithm

```
vector<pair<int, int>> g[MAX];
int cost[MAX] = { INITIALIZED BY INF };

void dijkstra(int start) {
  priority_queue<pair<int, int>> pq;
  pq.push(start, 0);

  while (!pq.empty()) {
    int now_v = pq.front().first;
    int now_cost = -pq.front().second;

    for (auto& i : g[now_v]) {
      int nxt_cost = i.second;

      if (cost[i] > nxt_cost) {
        cost[i] = nxt_cost;
        pq.push(make_pair(i.first, -nxt_cost));
      }
    }
  }
}
```

4.6 Floyd-Warshall Algorithm

```
int cost[MAX] = { INITIALIZED BY INF };

for (int k = 0; k < n; k++) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (cost[i][j] > cost[i][k] + cost[k][j])
        cost[i][j] = cost[i][k] + cost[k][j];
    }
  }
}
```