[일반 탐욕적 알고리즘]

```
#include <iostream>
#include <vector>

// 임의의 선택 작업을 수행하여 결과를 반환하는 함수
int select() {
    // 선택 작업을 수행하는 코드 작성
    // 이 예시에서는 임의의 정수를 반환하도록 구현하였습니다.
    return rand() % 10;
}

// 선택한 결과가 타당한지 여부를 판단하는 함수
bool isFeasible(int s) {
    // 타당성을 판단하는 코드 작성
    // 이 예시에서는 선택한 값이 5 이상인 경우에만 타당하다고 가정하였습니다.
    return s >= 5;
}

// 선택한 결과로 문제가 해결되었는지 여부를 판단하는 함수
bool isSolved(const std::vector<int>& S) {
    // 문제 해결 여부를 판단하는 코드 작성
    // 이 예시에서는 선택한 값의 합이 20 이상인 경우에만 문제가 해결되었다고 가정하였습니다.
    int sum = 0;
    for (int s : S) {
        sum += s;
    }
    return sum >= 20;
}

std::vector<int> greedy() {
    std::vector<int> S;
    while (true) {
        int s = select();
        if (isFeasible(s)) {
            S.push_back(s);
        }
        if (isSolved(S)) {
            break;
        }
    }
    return S;
}

int main() {
    std::vector<int> solution = greedy();
    std::cout << "Solution: ";
    for (int s : solution) {
        std::cout << s << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

[거스름 알고리즘]

```
#include <iostream>
#include <vector>

int coinChange(std::vector<int>& d, int change) {
    int count = 0;
    int n = d.size();
    int i = 0;
    while (i < n) {
        int coin = d[i];
        if (coin <= change) {
            count++;
            change -= coin;
            if (change == 0) break;
        }
        else i++;
    }
    return (change == 0) ? count : -1;
}

int main() {
    std::vector<int> coins = {1, 5, 10, 25};
    int change = 63;
    int minCoinCount = coinChange(coins, change);
    if (minCoinCount != -1)
        std::cout << "Minimum number of coins required: " << minCoinCount << std::endl;
    else std::cout << "No solution exists." << std::endl;
    }
    return 0;
}
```

[시스템 내부 총 시간 최소화 알고리즘]

```
#include <iostream>
#include <vector>
#include <algorithm>

int schedule(std::vector<int>& l) {
    std::sort(l.begin(), l.end()); // 작업 길이를 오름차순으로 정렬
    int total = 0;
    int c = 0;
    for (int i = 0; i < l.size(); i++) {
        c += l[i];
        total += c;
    }
    return total;
}

int main() {
    std::vector<int> jobLengths = {3, 1, 4, 2};
    int minTotalTime = schedule(jobLengths);
    std::cout << "Minimum total time: " << minTotalTime << std::endl;
    return 0;
}
```

[가중 완료 총 시간 최소화 알고리즘]

```
#include <iostream>
#include <vector>
#include <algorithm>

struct Job {
    int w;
    int l;
};

bool compareJobs(const Job& j1, const Job& j2) {
    return (j1.w * j2.l) > (j2.w * j1.l); // wi/li 기준으로 내림차순 정렬
}

int schedule(std::vector<Job>& J) {
    std::sort(J.begin(), J.end(), compareJobs); // 가중치 기준으로 정렬
    int total = 0;
    int c = 0;
    for (int i = 0; i < J.size(); i++) {
        c += J[i].l;
        total += J[i].w * c;
    }
    return total;
}

int main() {
    std::vector<Job> jobs = {{3, 4}, {1, 1}, {4, 2}, {2, 3}};
    int minTotalTime = schedule(jobs);
    std::cout << "Minimum total time: " << minTotalTime << std::endl;
    return 0;
}
```

[마감 시간이 있는 최적 스케줄링 알고리즘]

```
#include <iostream>
#include <vector>
#include <algorithm>

struct Job {
    int d;
    int w;
};

bool compareJobs(const Job& j1, const Job& j2) {
    return j1.d > j2.d; // 마감 시간을 기준으로 내림차순 정렬
}

std::vector<int> schedule(std::vector<Job>& J) {
    std::sort(J.begin(), J.end(), compareJobs); // 마감 시간을 기준으로 정렬
    std::vector<int> S = {1}; // 스케줄에 포함된 작업의 인덱스 집합
    for (int j = 1; j < J.size(); j++) {
        std::vector<int> K = S;
        K.push_back(j + 1); // 작업 인덱스는 1부터 시작하므로 j+1을 추가
        if (isFeasible(J, K)) {
            S = K;
        }
    }
    return S;
}

bool isFeasible(const std::vector<Job>& J, const std::vector<int>& K) {
    int maxDeadline = 0;
    for (int k : K) {
        maxDeadline = std::max(maxDeadline, J[k - 1].d); // 작업 인덱스는 1부터 시작하므로 k-1을 사용
        if (maxDeadline > k) {
            return false;
        }
    }
    return true;
}

int main() {
    std::vector<Job> jobs = {{4, 3}, {2, 2}, {3, 1}, {1, 4}};
    std::vector<int> optimalSchedule = schedule(jobs);
    std::cout << "Optimal Schedule: ";
    for (int j : optimalSchedule) {
        std::cout << j << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

[가장 큰 수 만들기]

```
#include <iostream>
#include <vector>
#include<algorithm>
using namespace std;

bool cmp(string a, string b) {
    return a + b > b + a;
}
void biggest(vector<string>& arr) {
    sort(arr.begin(), arr.end(), cmp);
    for (int i = 0; i < arr.size(); i++) cout << arr[i];
    cout << '\n';
}

int main() {
    int t, n;
    cin >> t;
    while (t--) {
        cin >> n;
        vector<string> arr(n);
        for (int i = 0; i < n; i++) cin >> arr[i];
        biggest(arr);
    } return 0;
}
```

[행성 충돌]

```
#include <iostream>
#include <vector>
#include<deque>
using namespace std;

void collide(int n) {
    deque<int> r;
    vector<int> ans;
    int a;
    for (int i = 0; i < n; i++) {
        cin >> a;
        // a가 음수
        if (a < 0) {
            if (r.empty()) ans.push_back(a);
            else {
                while (!r.empty()) {
                    if (r.back() > -a) break;
                    else if (r.back() == -a) {
                        r.pop_back();
```

```
                    break;
                }
                else {
                    r.pop_back();
                    if (r.empty())
                        ans.push_back(a);
                }
            }
        }
        // a가 양수
        else r.push_back(a);
    }
    for (int i = 0; i < ans.size(); i++) cout << ans[i] << ' ';
    while (!r.empty()) {
        cout << r.front() << ' ';
        r.pop_front();
    }
    cout << '\n';
}

int main() {
    int t, n, a;
    cin >> t;
    while (t--) {
        cin >> n;
        collide(n);
    }
    return 0;
}
```

[Prim 알고리즘]

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>

struct Edge {
    int u;
    int v;
    int weight;
};

struct CompareEdge {
    bool operator()(const Edge& e1, const Edge& e2) {
        return e1.weight > e2.weight; // 오름차순으로 정렬하기 위해 '>' 사용
    }
};

std::vector<Edge> prim(std::vector<std::vector<std::pair<int, int>>>& G, int start) {
    int V = G.size();
    std::vector<bool> visited(V, false); // 노드의 방문 여부를 나타내는 배열
    std::vector<Edge> T; // 최소 신장 트리의 간선들을 저장하는 배열
    std::priority_queue<Edge, std::vector<Edge>, CompareEdge> pq; // 가중치 오름차순으로 정렬된 간선들을 저장하는 우선순위 큐

    visited[start] = true;
    for (auto& edge : G[start]) {
        pq.push({start, edge.first, edge.second});
    }

    while (!pq.empty()) {
        Edge e = pq.top();
        pq.pop();

        if (visited[e.v]) {
            continue;
        }

        T.push_back(e);
        visited[e.v] = true;

        for (auto& edge : G[e.v]) {
            if (!visited[edge.first]) {
                pq.push({e.v, edge.first, edge.second});
            }
        }
    }

    return T;
}

int main() {
    int V = 5; // 노드의 개수
    std::vector<std::vector<std::pair<int, int>>> graph(V);

    // 간선 정보 추가
    graph[0].push_back({1, 2});
    graph[0].push_back({3, 6});
    graph[1].push_back({0, 2});
    graph[1].push_back({2, 3});
    graph[1].push_back({3, 8});
    graph[1].push_back({4, 5});
    graph[2].push_back({1, 3});
    graph[2].push_back({4, 7});
    graph[3].push_back({0, 6});
    graph[3].push_back({1, 8});
    graph[4].push_back({1, 5});
    graph[4].push_back({2, 7});

    int startNode = 0; // 시작 노드

    std::vector<Edge> minimumSpanningTree = prim(graph, startNode);

    std::cout << "Minimum Spanning Tree Edges: " << std::endl;
    for (const auto& edge : minimumSpanningTree) {
        std::cout << edge.u << " - " << edge.v << " : " << edge.weight << std::endl;
    }

    return 0;
}
```

[우선순위 큐 기반 Prim 알고리즘]

```
#include <iostream>
#include <vector>
#include <queue>
#include <utility>
#include <climits>

struct Edge {
    int u;
    int v;
    int weight;
};

struct CompareEdge {
    bool operator()(const Edge& e1, const Edge& e2) {
        return e1.weight > e2.weight; // 오름차순으로 정렬하기 위해 '>' 사용
    }
};

std::vector<Edge> prim(std::vector<std::vector<std::pair<int, int>>>& G, int start) {
    int V = G.size();
    std::vector<bool> visited(V, false); // 노드의 방문 여부를 나타내는 배열
    std::vector<Edge> T; // 최소 신장 트리의 간선들을 저장하는 배열
    std::vector<int> score(V, INT_MAX); // 각 노드의 현재 가중치
    std::priority_queue<Edge, std::vector<Edge>, CompareEdge> H; // 가중치 오름차순으로 정렬된 간선들을 저장하는 우선순위 큐

    visited[start] = true;
    score[start] = 0;
    for (auto& edge : G[start]) {
        int v = edge.first;
        int weight = edge.second;
        score[v] = weight;
        H.push({start, v, weight});
    }
    while (!H.empty()) {
        Edge e = H.top();
        H.pop();
        int u = e.v;

        if (visited[u]) continue;

        int v = e.u;
        visited[u] = true;
        T.push_back(e);

        for (auto& edge : G[u]) {
            int w = edge.first;
            int weight = edge.second;
            if (!visited[w] && weight < score[w]) {
                score[w] = weight;
                H.push({u, w, weight});
            }
        }
    }
    return T;
}

int main() {
    int V = 5; // 노드의 개수
    std::vector<std::vector<std::pair<int, int>>> graph(V);

    // 간선 정보 추가
    ~추가하기~
    int startNode = 0; // 시작 노드
    std::vector<Edge> minimumSpanningTree = prim(graph, startNode);
    std::cout << "Minimum Spanning Tree Edges: " << std::endl;
    for (const auto& edge : minimumSpanningTree) {
        std::cout << edge.u << " - " << edge.v << " : " << edge.weight << std::endl;
    }
    return 0;
}
```

[그래프 입력]

```
int main() {
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
    int t, n, e;
    int a, b, c;
    cin >> t;
    while (t--) {
        cin >> n >> e;
        vector<vector<pair<int, int>>> graph(n);
        for (int i = 0; i < e; i++) {
            cin >> a >> b >> c;
            graph[a].emplace_back(make_pair(b, c));
            graph[b].emplace_back(make_pair(a, c));
        }
        vector<int> commonNodes = findCommonEdges(graph, n);
        for (auto i : commonNodes) cout << i << ' ';
        cout << '\n';
    }
    return 0;
}
```

[가중치 무방향 그래프에서 최소신장트리 구하기 (Prime)]

```
#include <iostream>
#include <vector>
#include<queue>
#include<algorithm>

using namespace std;

struct Edge {
    int start, end, weight;
};

class UnionFind {
private:
    vector<int> parent, rank;

public:
    UnionFind(int n) {
```

```
        parent.resize(n + 1);
        rank.resize(n + 1);
        for (int i = 0; i <= n; i++) {
            parent[i] = i;
            rank[i] = 0;
        }
    }

    int find(int v) {
        if (parent[v] == v) return v;
        parent[v] = find(parent[v]);
        return parent[v];
    }

    void unite(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);

        if (rootX == rootY) {
            return;
        }

        if (rank[rootX] < rank[rootY]) {
            parent[rootX] = rootY;
        }
        else if (rank[rootX] > rank[rootY]) {
            parent[rootY] = rootX;
        }
        else {
            parent[rootY] = rootX;
            rank[rootX]++;
        }
    }
};

bool cmp(Edge a, Edge b) {
    return a.weight < b.weight;
}

int kruskal(vector<Edge>& edges) {
    int n = edges.size();
    vector<Edge> tree;
    UnionFind uf(n);

    sort(edges.begin(), edges.end(), cmp);

    for (Edge i : edges) {
        if (uf.find(i.start) != uf.find(i.end)) {
            tree.push_back(i);
            uf.unite(i.start, i.end);
        }
    }

    int ret = 0;
    for (Edge i : tree) {
        ret += i.weight;
    }
    return ret;
}

int main() {
    ios_base::sync_with_stdio(false);
cin.tie(NULL); cout.tie(NULL);

    int t, n, e;
    int a, b, c;
    cin >> t;
    while (t--) {
        cin >> n >> e;
        vector<Edge> edges(e);
        for (int i = 0; i < e; i++) {
            cin >> a >> b >> c;
            edges[i] = { a,b,c };
        }
        cout << kruskal(edges) << '\n';
    }
    return 0;
}
```

[피보나치 수열 (Memoization)]

```
int fibonacci(int n, std::vector<int>& memo) {
    if (n <= 1) return n;
    if (memo[n] != -1) return memo[n];

    memo[n] = fibonacci(n - 1, memo) +
fibonacci(n - 2, memo);
    return memo[n];
}
```

[피보나치 수열 (Memoization)]

```
int fibonacci(int n) {
    std::vector<int> table(n + 1, 0);
    table[1] = 1;
    for (int i = 2; i <= n; ++i)
        table[i] = table[i - 1] + table[i - 2];
    return table[n];
}
```

[0-1 배낭 tabulation]

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include<algorithm>

using namespace std;

struct Item {
    int weight;
    int value;
};

int knapsack(vector<Item>& items, int m) {
    int n = items.size();
    vector<vector<int>>      table(n      +      1,
vector<int>(m + 1, 0));

    for (int i = 1; i <= n; i++) {
        for (int x = 1; x <= m; x++) {
            if (items[i - 1].weight > x) {
                table[i][x] = table[i - 1][x];
            }
            else {
                table[i][x] = max(table[i - 1][x],
table[i - 1][x - items[i - 1].weight] + items[i -
1].value);
            }
        }
    }

    return table[n][m];
}

int main() {
    ios_base::sync_with_stdio(false);
cin.tie(NULL); cout.tie(NULL);

    int t, m, n;
    cin >> t;
    while (t--) {
        cin >> m >> n;
        vector<Item> arr(n);
        for (int i = 0; i < n; i++) {
            cin >> arr[i].value >> arr[i].weight;
        }
        cout << knapsack(arr, m) << '\n';
    }
    return 0;
}
```

[CanSum: large - 메모이제이션]

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include<algorithm>

using namespace std;

bool canSum(int m, const vector<int>& arr,
unordered_map<int, bool>& memo) {
    if (m < 0) return false;
    if (m == 0) return true;
    if (memo.count(m) > 0) return memo[m];
    for (int x : arr) {
        if (canSum(m - x, arr, memo)) {
            memo[m] = true;
            return true;
        }
    }
    memo[m] = false;
    return false;
}
bool canSum(int m, const vector<int>& arr) {
    unordered_map<int, bool> memo;
    return canSum(m, arr, memo);
}

int main() {
    ios_base::sync_with_stdio(false);
cin.tie(NULL); cout.tie(NULL);

    int t, m, n, a;
    cin >> t;
    while (t--) {
        cin >> m >> n;
        vector<int> arr(n);
        for (int i = 0; i < n; i++) {
            cin >> a;
            arr[i] = a;
        }
        if (canSum(m, arr)) cout << "true\n";
        else cout << "false\n";
    }
    return 0;
}
```

[CountSum: large - 메모이제이션]

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include<algorithm>

using namespace std;

long long countSum(int m, const vector<int>&
arr, unordered_map<int, long long>& memo) {
    if (m < 0) return 0;
    if (m == 0) return 1;
    if (memo.find(m) != memo.end()) return
memo[m];

    long long cnt = 0;
    for (int x : arr) {
        cnt += countSum(m - x, arr, memo);
    }
    memo[m] = cnt;
    return cnt;
}

int main() {
    ios_base::sync_with_stdio(false);
cin.tie(NULL); cout.tie(NULL);

    int t, m, n, a;
    cin >> t;
    while (t--) {
        cin >> m >> n;
        vector<int> arr(n);
        for (int i = 0; i < n; i++) {
            cin >> a;
            arr[i] = a;
        }
        unordered_map<int, long long> memo;
        cout << countSum(m, arr, memo) <<
'\n';
    }
    return 0;
}
```

[합계가 가장 큰 구간 찾기 테블레이션 알고리즘]

```
#include <iostream>
#include <vector>
#include <algorithm>
int maxSubsequence(std::vector<int>& A) {
    int n = A.size();
    int L = A[0];
    int S = L;
    for (int i = 1; i < n; ++i) {
        L = std::max(L + A[i], A[i]);
        S = std::max(S, L);
    }
    return S;
}
```

[최대 증가 부분 수열 찾기 메모이제이션]

```
int lis(const std::vector<int>& A) {
    int n = A.size();
    std::vector<int> memo(n, -1);
```

```
    int ret = 0;
    for (int i = 0; i < n; ++i)
        ret = std::max(lis(A, i, memo), ret);
    return ret;
}
int lis(const std::vector<int>& A, int curr, std::vector<int>& memo) {
    int n = A.size();
    if (memo[curr] != -1)
        return memo[curr];
    int ret = 1;
    for (int next = curr + 1; next < n; ++next) {
        if (A[curr] < A[next])
            ret = std::max(lis(A, next, memo) + 1, ret);
    }
    memo[curr] = ret;
    return memo[curr];
}
```

[유전자 염기서열 유사성 DP]

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int sequenceAlignment(string x, string y, int gapP, int misP) {
    int m = x.length();
    int n = y.length();

    vector<vector<int>> t(m + 1, vector<int>(n + 1, 0));
    for (int i = 1; i <= m; i++) t[i][0] = i * gapP;
    for (int i = 1; i <= n; i++) t[0][i] = i * gapP;
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            int p = x[i - 1] == y[j - 1] ? 0 : misP;
            t[i][j] = min({ p + t[i - 1][j - 1], gapP + t[i - 1][j], gapP + t[i][j - 1] });
        }
    }
    return t[m][n];
}

int main() {
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);

    int t, g, m;

    cin >> t;
    while (t--) {
        string x, y;
        cin >> g >> m >> x >> y;
        cout << sequenceAlignment(x, y, g, m) << '\n';
    }
    return 0;
}
```

[최적 BST 테블레이션 알고리즘]

```
double optimalBST(const std::vector<double>& p)
{
    int n = p.size() - 1;
    std::vector<std::vector<double>> table(n + 2, std::vector<double>(n + 2, 0));

    for (int s = 0; s < n; ++s) {
        for (int i = 1; i <= n - s; ++i) {
            double sumP = 0;
            double min = INT_MAX;

            for (int k = i; k <= i + s; ++k) {
                sumP += p[k];
                min = std::min(min, table[i][k - 1] + table[k + 1][i + s]);
            }

            table[i][i + s] = sumP + min;
        }
    }

    return table[1][n];
}
```

```
[ 목표합 찾기 DP ]
#include <iostream>
#include <vector>
#include <unordered_map>
#include <string>

using namespace std;

int countTargetSum(vector<int>& arr, int target, int index, int currSum, unordered_map<string, int>& memo) {
    string key = to_string(index) + "-" + to_string(currSum);

    if (memo.find(key) != memo.end()) return memo[key];

    if (index == arr.size()) {
        if (currSum == target) memo[key] = 1;
        else memo[key] = 0;
        return memo[key];
    }

    // 현재 인덱스의 원소를 더하는 경우
    int count1 = countTargetSum(arr, target, index + 1, currSum + arr[index], memo);

    // 현재 인덱스의 원소를 빼는 경우
    int cnt2 = countTargetSum(arr, target, index + 1, currSum - arr[index], memo);

    // 현재 경우의 수를 저장하고 반환
    memo[key] = count1 + cnt2;
    return memo[key];
}

int countTargetSum(vector<int>& arr, int target)
{
    unordered_map<string, int> memo;
    return countTargetSum(arr, target, 0, 0, memo);
}
```

```
[ 방향 그래프에서 최단 경로 찾기 ]
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

vector<vector<int>> floyd(vector<vector<int>>& graph) {
    int n = graph.size();
    vector<vector<int>> dist(n, vector<int>(n, 9876543));

    for (int i = 0; i < n; i++) {
        dist[i][i] = 0;
        for (int j = 0; j < n; j++) {
            if (graph[i][j] != 9876543)
                dist[i][j] = graph[i][j];
        }
    }

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dist[i][k] != 9876543 && dist[k][j] != 9876543)
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }

    for (int i = 0; i < n; i++) {
        if (dist[i][i] < 0)
            return vector<vector<int>>();
    }
    return dist;
}

int main() {
    int t, n, e;
    cin >> t;

    while (t-- > 0) {
        cin >> n >> e;

        vector<vector<int>> graph(n, vector<int>(n, 9876543));
        for (int i = 0; i < n; i++) graph[i][i] = 0;

        int start, end, w;
        for (int i = 0; i < e; i++) {
            cin >> start >> end >> w;
            graph[start][end] = w;
        }
        vector<vector<int>> dist = floyd(graph);
        if (dist.empty()) cout << -1 << endl;
        else {
            int d = -9876543;
            int y = 0, x = 0;
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    if (dist[i][j] <= 1000000 && dist[i][j] > d) {
                        d = dist[i][j];
                        y = i;
                        x = j;
                    }
                }
            }
            cout << y << " " << x << " " << d << endl;
        }
    }
    return 0;
}
```

[Bellman-Ford 알고리즘]

```
#include <iostream>
#include <vector>
#include <limits>
struct Edge {
    int source;
    int destination;
    int weight;
};

std::vector<int> bellmanFord(const std::vector<Edge>& edges, int n, int s) {
    std::vector<int> dist(n, std::numeric_limits<int>::max());
    dist[s] = 0;

    for (int i = 1; i <= n - 1; ++i) {
        bool found = false;
        for (const auto& edge : edges) {
            int u = edge.source;
            int v = edge.destination;
            int w = edge.weight;

            if (dist[u] != std::numeric_limits<int>::max() && dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                found = true;
            }
        }

        if (!found) break;
    }

    // 음의 주기 검사
    for (const auto& edge : edges) {
        int u = edge.source;
        int v = edge.destination;
        int w = edge.weight;

        if (dist[u] != std::numeric_limits<int>::max() && dist[u] + w < dist[v]) {
            return std::vector<int>();  // null 반환
        }
    }

    return dist;
}
```