Contents

1 Sync

1.1 Sync

```cpp
int main(){
    std::ios::sync_with_stdio(false);
    // 開始寫程式
}
```

2 Data Structure

2.1 Binary Search

```cpp
int binary_search(int arr[maxn], int lef, int rig,
    int target){
    if(lef > rig) return 0x3f3f3f3f;
    int mid = (lef + rig) >> 1;
    if(arr[mid] == target) return mid;
    else if(arr[mid] > target){
        return binary_search(arr, lef, mid - 1,
            target);
    }
    else{
        return binary_search(arr, mid + 1, rig,
            target);
    }
}
```

2.2 BIT

```cpp
#define lowbit(k) (k & -k)
void add(vector<int> &tr, int id, int val) {
  for (; id <= n; id += lowbit(id)) {
    tr[id] += val;
  }
}
int sum(vector<int> &tr, int id) {
  int ret = 0;
  for (; id >= 1; id -= lowbit(id)) {
    ret += tr[id];
  }
  return ret;
}
```

2.3 BWT

```cpp
/*BWT 資料轉換演算法*/
void BWT(){
    for(int i = 0; i < n; ++i){
        if(back[i] == 0)
            mini[zero++] = i;
    for(int i = 0; i < n; ++i)
        if(back[i] == 1)
            mini[zero++] = i;
    int ptr = mini[0];
    for(int i = 0; i < n; ++i){
        cout << back[ptr] << " ";
        ptr = mini[ptr];
    }
    cout << endl;
}
```

3 Divide and Conquer

3.1 count inversions

```cpp
/*逆序數對*/
int arr[maxn], buf[maxn];
int count_inversions(int lef, int rig){
    if(rig - lef <= 1) return 0;
    int mid = (lef + rig)/2;
    int ans = count_inversions(lef, mid) +
        count_inversions(mid, rig);
    int i = lef, j = mid, k = lef;
    while(i < mid || j < rig){
        if(i >= mid) buf[k] = arr[j++];
        else if(j >= rig) buf[k] = arr[i++];
```

```
        else{
            if(arr[i] <= arr[j]) buf[k] = arr[i++];
            else{
                buf[k] = arr[j++];
                ans += mid - i;
            }
        }
        k++;
    }
    for(int k = lef; k < rig; ++k) arr[k] = buf[k];
    return ans;
}
```

4 DP

4.1 Doubling

```
/* 倍增 */
int LOG = sqrt(N); // 2^LOG >= N
vector<int> arr(N);
vector<vector<int>> dp(N, vector<int>(LOG));
for(int i = 0; i < N; ++i) cin >> arr[i];
int L, Q, a, b;
cin >> L >> Q;
for(int i = 0; i < N; ++i){
    dp[i][0] = lower_bound(arr.begin(), arr.end(),
        arr[i] + L) - arr.begin();
    if(dp[i][0] == N || arr[i] + L < arr[dp[i][0]])
        dp[i][0] -= 1;
}
for(int i = 1; i < LOG; ++i)
    for(int j = 0; j < N; ++j)
        dp[j][i] = dp[dp[j][i - 1]][i - 1];
for(int i = 0; i < Q; ++i){
    cin >> a >> b;
    a--; // 要減減是因為arr的index從0開始但題目從1開始
    b--;
    if(a > b) swap(a, b);
    int ans = 0;
    for(int i = LOG - 1; i >= 0; --i){ // 從後往回推
        if(dp[a][i] < b){
            ans += (1 << i);
            a = dp[a][i];
        }
    }
    cout << ans + 1 << endl;
}
```

4.2 LCS

```
int LCS(string s1, string s2) {
  int n1 = s1.size(), n2 = s2.size();
  int dp[n1+1][n2+1] = {0};
  // dp[i][j] = s1的前i個字元和s2的前j個字元
  for (int i = 1; i <= n1; i++) {
    for (int j = 1; j <= n2; j++) {
      if (s1[i - 1] == s2[j - 1]) {
        dp[i][j] = dp[i - 1][j - 1] + 1;
      } else {
        dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
      }
    }
  }
  return dp[n1][n2];
}
```

4.3 LIS

```
int LIS(vector<int> &a) { // Longest Increasing
    Subsequence
  vector<int> s;
  for (int i = 0; i < a.size(); i++) {
    if (s.empty() || s.back() < a[i]) {
      s.push_back(a[i]);
    } else {
      *lower_bound(s.begin(), s.end(), a[i],
        [](int x, int y) {return x < y;}) = a[i];
    }
  }
  return s.size();
}
```

5 Enumerate

5.1 Halfcut Enumerate

```
/* 折半枚舉 */
void dfs(set<long long int> &s, int depth, int T,
    long long int sum){
    if(depth >= T){
        s.insert(sum);
        return;
    }
    dfs(s, depth + 1, T, sum); // 取或不取的概念
    dfs(s, depth + 1, T, sum + A[depth]);
}
int main(){
    int N, T;
    set<long long int> s1, s2;
    cin >> N >> T;
    for(int i = 0; i < N; ++i) cin >> A[i];
    dfs(s1, 0, N/2, 0); // 折半枚舉
    dfs(s2, N/2, N, 0);
    long long int ans = 0;
    // 題目:枚舉集合 Sx 的數字 Sxi，找出 Sy
        集合內小於等於 T-Sxi 中最大的數 Syj
    for(auto &x : s1){
        auto it = s2.upper_bound(T - x);
        long long int y = *(--it);
        if(x + y <= T) ans = max(ans, x + y);
    }
    cout << ans << endl;
}
```

6 Graph

6.1 SPFA

```
bool SPFA(int s){
    // 記得初始化這些陣列
    int cnt[1000+5], dis[1000+5];
    bool inqueue[1000+5];
    queue<int> q;

    q.push(s);
    dis[s] = 0;
    inqueue[s] = true;
    cnt[s] = 1;
    while(!q.empty()){
        int now = q.front();
        q.pop();
        inqueue[now] = false;

        for(auto &e : G[now]){
            if(dis[e.t] > dis[now] + e.w){
                dis[e.t] = dis[now] + e.w;
                if(!inqueue[e.t]){
                    cnt[e.t]++;
```

```
                    if(cnt[e.t] > m){
                        return false;
                    }
                    inqueue[e.t] = true;
                    q.push(e.t);
                }
            }
        }
    }
    return true;
}
```

6.2 Dijkstra

```
struct Item{
    int u, dis;
    // 取路徑最短
    bool operator < (const Item &other) const{
        return dis > other.dis;
    }
};
int dis[maxn];
vector<Edge> G[maxn];
void dijkstra(int s){
    for(int i = 0; i <= n; i++){
        dis[i] = inf;
    }
    dis[s] = 0;
    priority_queue<Item> pq;
    pq.push({s, 0});
    while(!pq.empty()){
        // 取路徑最短的點
        Item now = pq.top();
        pq.pop();
        if(now.dis > dis[now.u]){
            continue;
        }
        // 鬆弛更新，把與 now.u 相連的點都跑一遍
        for(Edge e : G[now.u]){
            if(dis[e.v] > now.dis + e.w){
                dis[e.v] = now.dis + e.w;
                pq.push({e.v, dis[e.v]});
            }
        }
    }
}
```

6.3 Floyd Warshall

```
void floyd_warshall(){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            G[i][j] = INF;
        }
        G[i][i] = 0;
    }
    for (int k = 0; k < n; k++){      // 嘗試每一個中繼點
        for (int i = 0; i < n; i++){ // 計算每一個i點與每一個j點
            for (int j = 0; j < n; j++){
                G[i][j] = min(G[i][j], G[i][k] +
                    G[k][j]);
            }
        }
    }
}
```

6.4 Disjoint set Kruskal

```
struct Edge{
    int u, v, w;
    // 用權重排序 由大到小
    bool operator < (const Edge &other) const{
        return w > other.w;
    }
}edge[maxn];
// disjoint set
int find(int x){
  if(parent[x] < 0){
    return x;
  }
  else{
    return parent[x] = find(parent[x]);
  }
}
void unite(int a, int b){
  a = find(a);
  b = find(b);

  if(a != b){
    if(parent[a] < parent[b]){
      parent[a] += parent[b];
      parent[b] = a;
    }
    else{
      parent[b] += parent[a];
      parent[a] = b;
    }
  }
}
void kruskal(){
    memset(parent, -1, sizeof(parent));
    sort(edge, edge + m);
    int i, j;
    for(i = 0, j = 0; i < n - 1 && j < m; i++){
        // 如果 u 和 v 的祖先相同，則 j++
        (祖先相同代表會產生環 所以不要)
        while(find(edge[j].u) == find(edge[j].v)) j++;
        // 若部會產生環 則讓兩點之間產生橋
        (連接兩顆子生成樹)
        unite(edge[j].u, edge[j].v);
        j++;
    }
}
```

6.5 Bipatirate

```
/* 二分圖 */
const int maxn = 300 + 5;
int n, color[maxn];
vector<vector<int>> v(maxn);
bool dfs(int s){
    for(auto it : v[s]){
        if(color[it] == -1){
            color[it] = 3 - color[s];
            if(!dfs(it)){
                return false;
            }
        }
        if(color[s] == color[it]){
            return false;
        }
    }
    return true;
}
void isBipatirate(){
    bool flag = true;
    for(int i = 1; i <= n; ++i){
        if(color[i] == -1){
            color[i] = 1;
            flag &= dfs(i);
        }
    }
    if(flag){
```

```
        cout << "YES" << endl;
    }
    else{
        cout << "NO" << endl;
    }
}
int main(){
    while(cin >> n && n){
        for(int i = 1; i <= n; ++i) v[i].clear();
        memset(color, -1, sizeof(color));
        int a, b;
        while(cin >> a >> b && (a || b)){
            v[a].emplace_back(b);
            v[b].emplace_back(a);
        }
        isBipatirate();
    }
}
```

6.6 Hungarian algorithm

```
/* 匈牙利演算法 */
const int maxn = 500+5;
int t, N, bn, gn, match[maxn];
bool visited[maxn];
vector<vector<int>> G(maxn);
struct People{
    int h;
    string music, sport;
    People(){}
    People(int h, string music, string sport){
        this->h = h;
        this->music = music;
        this->sport = sport;
    }
}lef[maxn], rig[maxn];
bool check(People boy, People girl){
    if(abs(boy.h - girl.h) <= 40 && boy.music ==
        girl.music && boy.sport != girl.sport) return
        true;
    return false;
}
bool dfs(int s){
    for(int i = 0; i < G[s].size(); ++i){
        int v = G[s][i];
        if(visited[v]) continue;
        visited[v] = true;
        if(match[v] == -1 || dfs(match[v])){
            match[v] = s;
            return true;
        }
    }
    return false;
}
int Hungarian(){
    int cnt = 0;
    memset(match, -1, sizeof(match));
    for(int i = 0; i < bn; ++i){
        memset(visited, false, sizeof(visited));
        if(dfs(i)) cnt++;
    }
    return cnt;
}
int main(){
    cin >> t;
    while(t--){
        cin >> N;
        bn = 0, gn = 0;
        for(int i = 0; i <= N; ++i) G[i].clear();
        int h;
        string sex, music, sport;
        for(int i = 0; i < N; ++i){
            cin >> h >> sex >> music >> sport;
            if(sex == "M") lef[bn++] = People(h,
                music, sport);
            else rig[gn++] = People(h, music, sport);
        }
        for(int i = 0; i < bn; ++i){
            for(int j = 0; j < gn; ++j)
                if(check(lef[i], rig[j]))
                    G[i].emplace_back(j);
        }
        cout << N - Hungarian() << endl;
    }
}
```

6.7 LCA

```
/*最低共同祖先*/
// 此 node 下有機顆 node
int dfs(int node, int dep){
    depth[node] = dep + 1;
    if(G[node].empty()){
        siz[node] = 1;
        return 1;
    }
    int total = 1;
    for(auto i : G[node])
        total += dfs(i.v, dep + 1);
    siz[node] = total;
    return siz[node];
}
// 找出每個節點的 2^i 倍祖先
// 2^20 = 1e6 > 200000
void find_parent(){
    for(int i = 1; i < 20; i++)
        for (int j = 0; j < N; j++)
            parent[j][i] =
                parent[parent[j][i-1]][i-1];
}
// 求兩點的LCA (利用倍增法)
int LCA(int a, int b){
    if (depth[b] < depth[a]) swap(a, b);
    if (depth[a] != depth[b]){
        int dif = depth[b] - depth[a];
        for (int i = 0; i < 20; i++){
            if (dif & 1) b = parent[b][i];
            dif >>= 1;
        }
    }
    if (a == b) return a;
    for (int i = 19; i >= 0; i--){
        if (parent[a][i] != parent[b][i]){
            a = parent[a][i];
            b = parent[b][i];
        }
    }
    return parent[a][0];
}
```

7 Other

7.1 Ants Colony

```
/* LCA 最低共同祖先 */
const int maxn = 1e5 + 5;
struct Edge{
    int v;
    int w;
};
int N;
vector<Edge> G[maxn];
int parent[maxn][20+5];
int depth[maxn], siz[maxn];
// 此 node 下有機顆 node
int dfs(int node, int dep){
    depth[node] = dep + 1;
    if(G[node].empty()){
```

```
        siz[node] = 1;
        return 1;
    }
    int total = 1;
    for(auto i : G[node])
        total += dfs(i.v, dep + 1);
    siz[node] = total;
    return siz[node];
}
// 找出每個節點的 2^i 倍祖先
// 2^20 = 1e6 > 200000
void find_parent(){
    for(int i = 1; i < 20; i++)
        for (int j = 0; j < N; j++)
            parent[j][i] =
                parent[parent[j][i-1]][i-1];
}
// 求兩點的LCA（利用倍增法）
int LCA(int a, int b){
    if (depth[b] < depth[a]) swap(a, b);
    if (depth[a] != depth[b]){
        int dif = depth[b] - depth[a];
        for (int i = 0; i < 20; i++){
            if (dif & 1) b = parent[b][i];
            dif >>= 1;
        }
    }
    if (a == b) return a;
    for (int i = 19; i >= 0; i--){
        if (parent[a][i] != parent[b][i]){
            a = parent[a][i];
            b = parent[b][i];
        }
    }
    return parent[a][0];
}
long long int dist[maxn];
// 從 0 開始到每個點的距離
void distance(){
    for (int u = 0; u < N; ++u){
        for(int i = 0; i < G[u].size(); ++i){
            dist[G[u][i].v] = dist[u] + G[u][i].w;
}
int main(){
    while(cin >> N && N){
        memset(dist, 0, sizeof(dist));
        memset(parent, 0, sizeof(parent));
        memset(depth, 0, sizeof(depth));
        memset(siz, 0, sizeof(siz));
        for(int i = 0; i <= N; ++i){
            G[i].clear();
        }
        for(int i = 1; i < N; ++i){
            int u, w;
            cin >> u >> w;
            G[u].push_back({i, w});
            parent[i][0] = u;
        }
        find_parent();
        dfs(0, 0);
        distance();
        int s; cin >> s;
        bool space = false;
        for(int i = 0; i < s; ++i){
            int a, b;
            cin >> a >> b;
            int lca = LCA(a, b);
            if(space) cout << " ";
            space = true;
            cout << (dist[a] + dist[b]) - (dist[lca]
                * 2);
        }
        cout << endl;
    }
}
```

7.2 Binary codes

```
/* BWT 資料轉換演算法 */
void BWT(){
    for(int i = 0; i < n; ++i){
        if(back[i] == 0){
            mini[zero++] = i;
    for(int i = 0; i < n; ++i){
        if(back[i] == 1){
            mini[zero++] = i;
    int ptr = mini[0];
    for(int i = 0; i < n; ++i){
        cout << back[ptr] << " ";
        ptr = mini[ptr];
    }
    cout << endl;
}
int main(){
    cin >> n;
    for(int i = 0; i < n; ++i){
        cin >> back[i];
    zero = 0;
    BWT();
}
```

7.3 Fire Fire Fire

```
/* dfs
只要我有一個小孩不是防火牆，我就必須是防火牆 */
const int maxn = 1000+5;
int cnt = 0;
vector<int> G[maxn];
bool exi[maxn], visited[maxn];
void dfs(int node, int parent){
    if(G[node].size() == 1 && G[node][0] == parent)
        return;
    for(int i = 0; i < G[node].size(); ++i){
        int now = G[node][i];
        if(visited[now]) continue;
        visited[now] = true;
        dfs(G[node][i], node);
    }
    bool flag = false;
    for(int j = 0; j < G[node].size(); ++j){
        if(exi[G[node][j]] != true && G[node][j] !=
            parent){
            flag = true;
            break;
        }
    }
    if(flag && exi[node] != true){
        exi[node] = true;
        cnt++;
    }
    return;
}
int main(){
    int n;
    while(cin >> n && n){
        for(int i = 1; i <= n; ++i) G[i].clear();
        memset(exi, false, sizeof(exi));
        memset(visited, false, sizeof(visited));
        for(int i = 1; i <= n; ++i){
            int siz; cin >> siz;
            for(int j = 0; j < siz; ++j){
                int num; cin >> num;
                G[i].emplace_back(num);
            }
        }
        cnt = 0;
        dfs(1, 1);
        if(n == 1) cnt++;
        cout << cnt << endl;
    }
}
```

8 DP

8.1 Crested Ibis vs Monster

```
/* dp 背包 - 重量/價值/可重複使用
9 3
8 3
4 2
2 1
0 3 3 3 3 3 3 3 3 6
0 2 2 2 2 3 3 3 3 5
0 1 1 2 2 3 3 3 3 4
因為這題可以重複使用同一條魔法
所以可以這樣dp */
int a[10000+5], b[10000+5];
int dp[10000+5][10000+5];
int main(){
    int h, n;
    cin >> h >> n;
    for(int i = 1; i <= n; i++)
        cin >> a[i] >> b[i];
    memset(dp, 0x3f3f3f3f, sizeof(dp));
    dp[0][0] = 0;
    for(int i = 1; i <= n; i++)
        for(int j = 0; j <= h; j++)
            dp[i][j] = min(dp[i-1][j], dp[i][max(0, j
                - a[i])] + b[i]);
    cout << dp[n][h] << endl;
}
```

8.2 dpd Knapsack 1

```
/* dp 背包 - 時間/數量/價值 - 第幾分鐘符合
w[i]: 3
陣列每一格代表的意義是最大上限為 index
    時可以放入的最大 value
0 0 0 30 30 30 30 30 30
w[i]: 4
0 0 0 30 50 50 50 80 80
w[i]: 5
0 0 0 30 50 60 60 80 90  */
int main(){
    int N, W;
    cin >> N >> W;
    int w[100000+5], v[100000+5];
    for(int i = 0; i < N; i++)
        cin >> w[i] >> v[i];
    long long int dp[100000+5];
    memset(dp, 0, sizeof(dp));
    for(int i = 0; i < N; i++)
        for(int j = W; j >= w[i]; j--)
            dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
    cout << dp[W] << endl;
}
```

8.3 Homer Simpson

```
/* dp 背包 - 時間/數量 - 漢堡
3 5 54
吃 3 分鐘漢堡時
0 -1 -1 1 -1 -1 2 -1 -1 3 -1 -1 4 -1 -1 5 -1 -1 6 -1
    -1 7 -1 -1 8 -1 -1 9 -1 -1 10 -1 -1 11 -1 -1 12
    -1 -1 13 -1 -1 14 -1 -1 15 -1 -1 16 -1 -1 17 -1
    -1 18
吃 5 分鐘漢堡時 (更新)
0 -1 -1 1 -1 1 2 -1 2 3 2 3 4 3 4 5 4 5 6 5 6 7 6 7 8
    7 8 9 8 9 10 9 10 11 10 11 12 11 12 13 12 13 14
    13 14 15 14 15 16 15 16 17 16 17 18
只有當該時間可剛好吃滿漢堡時會更新
全部初始設 -1，用以判斷 譬如當 1 分鐘時
    吃不了任何漢堡*/
int main(){
    int m, n, t;
    while(cin >> m >> n >> t){
        int dp[10000+5];
        memset(dp, -1, sizeof(dp));
        dp[0] = 0;
        for(int i = m; i <= t; i++)
            if(dp[i - m] != -1)
                dp[i] = max(dp[i], dp[i - m] + 1);
        for(int i = n; i <= t; i++)
            if(dp[i - n] != -1)
                dp[i] = max(dp[i], dp[i - n] + 1);
        // 時間無法剛好吃滿的時候
        if(dp[t] == -1){
            for(int i = t; i >= 0; i--)
                if(dp[i] != -1){
                    cout << dp[i] << " " << t - i <<
                        endl;
                    break;
                }
        }
        else cout << dp[t] << endl;
    }
}
```

8.4 Let Me Count The Ways

```
/* dp - 時間/數量 - 硬幣排序
要湊出 17
1 1 1 1 1 2 2 2 2 2 4 4 4 4 4 6 6 */
int main(){
    long long int n;
    long long int dp[30000+5];
    int coin[] = {1, 5, 10, 25, 50};
    memset(dp, 0, sizeof(dp));
    // 直接把 dp 做好
    dp[0] = 1;
    for(int i = 0; i < 5; i++)
        for(int j = coin[i]; j < 30000+5; j++)
            if(dp[j - coin[i]] != -1)
                dp[j] += dp[j - coin[i]];
    while(cin >> n){
        if(dp[n] == 1)
            cout << "There is only " << dp[n] << "
                way to produce " << n << " cents
                change." << endl;
        else
            cout << "There are " << dp[n] << " ways
                to produce " << n << " cents change."
                << endl;
    }
}
```

8.5 Luggage

```
/* dp 背包 - 重量/是否成立
7 7 13 1
1 1 0 0 0 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 0 0 0 0 0
    1
Note: dp[0] = true */
int main(){
    int t;
    cin >> t;
    cin.ignore();
    while(t--){
        string str;
        getline(cin , str);
        vector<int> v;
        stringstream ss;
        int num, cnt = 0, sum = 0;;
        bool dp[4000+5];
        memset(dp, false, sizeof(dp));
        ss << str;
```

```
        while(ss >> num){
            cnt++;
            sum += num;
            v.emplace_back(num);
        }
        if(sum & 1){
            cout << "NO" << endl;
            continue;
        }
        dp[0] = true;
        for(int i = 0; i < v.size(); i++)
            for(int j = sum; j >= v[i]; j--)
                if(dp[j - v[i]])
                    dp[j] = true;
        cout << (dp[sum/2] ? "YES" : "NO") << endl;
    }
}
```

8.6 Partitioning by Palindromes

```
/* string & dp - 字串長度判斷迴文
r a c e c a r
i = 0, j = 0
-> r = r ，dp[1] = dp[0] + 1 = 1
i = 1, j = 0
-> 因 a != r，dp[2] = 0x3f3f3f3f
i = 1, j = 1
-> 因 a = a, dp[2] = dp[1] + 1 = 2 */
bool check_palindromes(int lef, int rig){
    // 比較字串兩端都是迴文
    while(lef < rig){
        if(str[lef] != str[rig]) return 0;
        lef++;
        rig--;
    }
    return 1;
}
int main(){
    int t;
    cin >> t;
    while(t--){
        cin >> str;
        memset(dp, 0x3f3f3f3f, sizeof(dp));
        dp[0] = 0;
        for(int i = 0; i < str.size(); ++i)
            for(int j = 0; j <= i; ++j)
                if(str[i] == str[j])
                    if(check_palindromes(j, i))
                        if(dp[i+1] > dp[j] + 1)
                            dp[i+1] = dp[j] + 1;
        cout << dp[str.size()] << endl;
    }
}
```

8.7 SuperSale

```
/* dp 背包 - 重量/價值/不可重複使用
第一個人的負重: 23
0 0 0 0 52 52 52 52 52 54 54 54 54 106 106 106 106
    106 106 106 106 106 151 151
第二個人的負重: 20
0 0 0 0 52 52 52 52 52 54 54 54 54 106 106 106 106
    106 106 106 106
第三個人的負重: 20
0 0 0 0 52 52 52 52 52 54 54 54 54 106 106 106 106
    106 106 106 106
第四個人的負重: 26
0 0 0 0 52 52 52 52 52 54 54 54 54 106 106 106 106
    106 106 106 106 106 151 151 151 151 151 */
struct Edge{
    int p;
    int w;
}edge[1000+5];
int main(){
    int t;
    cin >> t;
    while(t--){
        int n; cin >> n;
        for(int i = 0; i < n; i++)
            cin >> edge[i].p >> edge[i].w;
        int g, total = 0;
        cin >> g;
        for(int i = 0; i < g; i++){
            int pw; in >> pw;
            int dp[30+5];
            memset(dp, 0, sizeof(dp));
            for(int j = 0; j < n; j++)
                for(int k = pw; k >= edge[j].w; k--)
                    dp[k] = max(dp[k], dp[k -
                        edge[j].w] + edge[j].p);
            total += dp[pw];
        }
        cout << total << endl;
    }
}
```

8.8 Walking on the Safe Side

```
/* dp - 地圖更新
更新地圖
一張如下的地圖 其 dp 更新方法為加上和加左的路
0 0 0 0 0
0 1 0 0 0
0 0 1 0 1
0 0 0 0 0
1 1 1 1 1
1 0 1 2 3
1 1 0 2 0
1 2 2 4 4 */
bool mp[100+5][100+5];
long long int dp[100+5][100+5];
int main(){
    int t; cin >> t;
    bool space = false;
    while(t--){
        if(space) cout << endl;
        else space = true;
        int r, c; cin >> r >> c;
        cin.ignore();
        memset(mp, false, sizeof(mp));
        memset(dp, 0, sizeof(dp));
        string str;
        for(int i = 0; i < r; i++){
            getline(cin, str);
            int n, num;
            stringstream ss(str);
            ss >> n;
            while(ss >> num)
                mp[n][num] = true;
        }
        dp[1][1] = 1;
        for(int i = 1; i <= r; i++){
            for(int j = 1; j <= c; j++){
                if(mp[i][j]) continue;
                if(i > 1)
                    dp[i][j] += dp[i-1][j];
                if(j > 1)
                    dp[i][j] += dp[i][j-1];
            }
        }
        cout << dp[r][c] << endl;
    }
}
```

8.9 Cutting Sticks

```cpp
/* dp - 動態切割取最小
100
3
25 50 75
dp:
0 0 50 125 200
0 0 0 50 125
0 0 0 0 50
0 0 0 0 0
0 0 0 0 0 */
int main(){
    int l;
    while(cin >> l && l){
        int n;
        cin >> n;
        vector<int> s(n+2);
        s[0] = 0;
        for(int i = 1; i <= n; ++i)
            cin >> s[i];
        // 從現在開始 n 的數量變為 n + 1
        s[++n] = l;
        int dp[n+5][n+5];
        memset(dp, 0, sizeof(dp));
        // r: 切幾段 b: 起點 c: 中間點 e: 終點
        for(int r = 2; r <= n; ++r){
            for(int b = 0; b < n; ++b){
                // 如果從 b 開始切 r 刀會超出長度就 break
                if(b + r > n) break;
                // e: 從 b 開始切 r 刀
                int e = b + r;
                dp[b][e] = 0x3f3f3f3f;
                // c: 遍歷所有從 b 開始到 e 結束的中間點
                for(int c = b + 1; c < e; ++c){
                    // dp[b][c] 從 b 到 c 最少 cost + dp[c][e] 從 c 到 e 最少 cost
                    // s[e] - s[b] 兩段之間的 cost
                    dp[b][e] = min(dp[b][e], dp[b][c] + dp[c][e] + s[e] - s[b]);
                }
            }
        }
        cout << "The minimum cutting is " << dp[0][n] << "." << endl;
    }
}
```

8.10 Race to 1

```cpp
/* dp - 數量
期望值、質數、dfs */
const int N = 1000000;
bool sieve[N+5];
vector<int> pri;
double dp[N+5];
// 線性篩
void Linear_Sieve(){
    for (int i = 2; i < N; i++){
        if (!sieve[i])
            pri.push_back(i);
        for (int p: pri){
            if (i * p >= N) break;
            sieve[i * p] = true;
            if (i % p == 0) break;
        }
    }
}
double dfs(int n){
    if(dp[n] != -1) return dp[n];
    dp[n] = 0;
    if(n == 1) return dp[n];
    int total = 0, prime = 0;
    for(int i = 0; i < pri.size() && pri[i] <= n; i++){
        total++;
        if(n % pri[i]) continue;
        prime++;
        dp[n] += dfs(n/pri[i]);
    }
    // 算期望值
    dp[n] = (dp[n] + total)/prime;
    return dp[n];
}
int main(){
    int t, num, ca = 1;
    for(int i = 0; i <= N; i++)
        dp[i] = -1;
    Linear_Sieve();
    cin >> t;
    while(t--){
        cin >> num;
        cout << "Case " << ca++ << ": " << fixed << setprecision(10) << dfs(num) << endl;
    }
}
```

9 Math

9.1 Big Mod

```python
'''
Mod
pow(x, y, z) = x^y % z
'''
# python 如何讀取直到 EOF 用 try except
try:
    while True:
        # input().split() 用空格切開讀取一整行
        # map (型態, input().split()) 才能把值全讀成 int
        B, P, M = map(int, input().split())
        print(pow(B, P, M))
except EOFError:
    exit
```

9.2 Bubble Sort Expect Value

```cpp
/* 數論 期望值算法:
擲一枚公平的六面骰子，其每次「點數」的期望值是 3.5
E(x) = 1 * 1/6 + 2 * 1/6 + 3 * 1/6 + 4 * 1/6 + 5 * 1/6 + 6 * 1/6
= (1 + 2 + 3 + 4 + 5 + 6)/6 = 3.5
bubble sort 每兩兩之間交換機率是 1/2
總共會做 C(n, 2) 次
E(x) = C(n, 2) * 1/2 = (n * (n - 1))/2 * 1/2 */
int t, ca = 1;
cin >> t;
while(t--){
    long long int n;
    cin >> n;
    cout << "Case " << ca++ << ": ";
    // 如果 (n * (n - 1)) 可以被 4 整除
    // 代表最後答案會是整數，否則會是分數
    if((n * (n - 1)) % 4){
        cout << ( (n * (n - 1)) / 2 ) << "/2"<< endl;
    }
    else{
        cout << ( (n * (n - 1)) / 2 ) / 2 << endl;
    }
}
```

9.3 Fraction Floor Sum

```cpp
/* 數論
[N/i] == M
-> M <= N/i < M + 1
-> N/(M+1) < i <= N/M */
int main(){
    long long int N;
    cin >> N;
    long long int ans = 0;
    for(long long int i = 1; i <= N; i++){
        long long int M = N / i, n = N / M;
        // 總共會有 n - i 個的 [N/i] 值都是 M
        ans += (n - i + 1) * M;
        // 更新跳過 以免重複計算
        i = n;
    }
    cout << ans << endl;
}
```

9.4 How Many Os

```cpp
/* 數論 */
int main(){
    long long int n, m;
    while(cin >> n >> m && (n >= 0) && (m >= 0)){
        long long int total1 = 0, total2 = 0;
        long long int ten = 1, tmp = n-1;
        while(tmp >= 10){
            if(tmp % 10 == 0){
                tmp /= 10;
                total1 += (tmp - 1) * ten + ((n-1) %
                    ten) + 1;
            }
            else{
                tmp /= 10;
                total1 += tmp * ten;
            }
            ten *= 10;
        }
        ten = 1; tmp = m;
        while(tmp >= 10){
            if(tmp % 10 == 0){
                tmp /= 10;
                total2 += (tmp - 1) * ten + (m % ten)
                    + 1;
            }
            else{
                tmp /= 10;
                total2 += tmp * ten;
            }
            ten *= 10;
        }
        if(n == 0) total1--;
        cout << total2 - total1 << endl;
    }
}
```

9.5 Number of Pairs

```cpp
/* 數論
uper_bound ex:
10 20 30 30 40 50
upper_bound for element 30 is at index 4
lower_bound ex:
10 20 30 40 50
lower_bound for element 30 at index 2 */
int main(){
    int t;
    cin >> t;
    while(t--){
        int n, l, r;
        vector<int> v;
        cin >> n >> l >> r;
        int num;
        for(int i = 0; i < n; i++){
            cin >> num;
            v.emplace_back(num);
        }
        sort(v.begin(), v.end());
        long long int ans = 0;
        for(int i = 0; i < n; i++)
            ans += (upper_bound(v.begin() + i + 1,
                v.end(), r - v[i]) -
                lower_bound(v.begin() + i + 1,
                v.end(), l - v[i]));
        cout << ans << endl;
    }
}
```

9.6 ORXOR

```cpp
/* bitwise operator 二進位制數論
如何切區段，之所以要1<<n是為了可以跑000~111
i = 0，binary i = 000
0 : 1 5 7
i = 1，binary i = 001
1 : 1 5 7
i = 2，binary i = 010，看得出來切了一刀
2 : 1 | 5 7
i = 3，binary i = 011
3 : 1 | 5 7
i = 4，binary i = 100，為了要切在index=2，所以才要1<<j
4 : 1 5 | 7
i = 5，binary i = 101
5 : 1 5 | 7
i = 6，binary i = 110
6 : 1 | 5 | 7
i = 7，binary i = 111
7 : 1 | 5 | 7
可以觀察出來，前兩位 bit 是 1 時代表的意義是切在哪裡
    */
int main(){
    int n; cin >> n;
    int num[20+7];
    memset(num, 0, sizeof(num));
    for(int i = 1; i <= n; i++)
        cin >> num[i];
    // 不知道為甚麼只有 2147483647 給過
    int mini = 2147483647;
    // 1 << n = n * 2
    for(int i = 0; i < (1 << n); i++){
        int XOR = 0, OR = 0;
        for(int j = 1; j <= n; j++){
            OR |= num[j];
            if((i & (1 << j))){
                XOR ^= OR;
                OR = 0;
            }
        }
        XOR ^= OR;
        mini = min(mini, XOR);
    }
    cout << mini << endl;
}
```

9.7 X drawing

```cpp
/* 數論畫圖 */
int main(){
    long long int n;
    long long int a, b;
    long long int p, q, r, s;
    cin >> n >> a >> b;
    cin >> p >> q >> r >> s;
    for(long long int i = p; i <= q; i++){
```

```
        for(long long int j = r; j <= s; j++)
            if(abs(i - a) == abs(j - b)) cout << '#';
            else cout << '.';
        cout << endl;
    }
}
```

10 Binary Search

10.1 Fill the Containers

```
/*binary search 變形*/
int binary_search(int arr[maxn], int lef, int rig,
    int mini){
    if(lef > rig) return mini;
    int amount = 1, fill = 0;
    int mid = (lef + rig) >> 1;
    for(int i = 0; i < n; ++i){
        if(amount > m) break;
        fill += arr[i];
        if(fill > mid){
            fill = arr[i];
            amount++;
        }
    }
    if(!flag && amount <= m) mini = mid;
    if(flag && amount == m) mini = mid;
    if(amount == m){
        flag = true;
        return binary_search(arr, lef, mid - 1, mid);
    }
    else if(amount < m){
        return binary_search(arr, lef, mid - 1, mini);
    }
    else{
        return binary_search(arr, mid + 1, rig, mini);
    }
}
int main(){
    int ca = 1;
    while(cin >> n >> m){
        flag = false;
        int arr[maxn];
        int maxi = 0, sum = 0;
        for(int i = 0; i < n; ++i){
            cin >> arr[i];
            sum += arr[i];
            maxi = max(maxi, arr[i]);
        }
        cout << binary_search(arr, maxi, sum, maxi)
            << endl;
    }
}
```

10.2 Where is the marble

```
/*upper_bound & lower_bound*/
int main(){
    int N, Q;
    int ca = 1;
    while(cin >> N >> Q && N && Q){
        vector<int> v(N);
        for(int i = 0; i < N; ++i) cin >> v[i];
        sort(v.begin(), v.end());
        cout << "CASE# " << ca++ << ":" << endl;
        int marble;
        for(int i = 0; i < Q; ++i){
            cin >> marble;
            int lef = lower_bound(v.begin(), v.end(),
                marble) - v.begin();
            int rig = upper_bound(v.begin(), v.end(),
                marble) - v.begin();
            if(lef == rig) cout << marble << " not
                found" << endl;
            else{
                cout << marble << " found at " << lef
                    + 1 << endl;
            }
        }
    }
}
```

11 Segement Tree

11.1 Frequent values

```
/* Segement Tree & RMQ (Range Sum Query)
idx:  1   2   3   4   5   6   7   8   9  10
num: -1  -1   1   1   1   1   3  10  10  10
fre:  2   2   4   4   4   4   1   3   3   3
border
left: 1   1   3   3   3   3   7   8   8   8
right:2   2   6   6   6   6   7  10  10  10 */
# define Lson(x) x << 1
# define Rson(x) (x << 1) + 1
const int maxn = 1e5+5;
struct Tree{
    int lef, rig, value;
}tree[4 * maxn];
struct Num{
    int lef, rig, value, fre;
}num[maxn];
// 建立 segement tree
void build(int lef, int rig, int x){
    tree[x].lef = lef;
    tree[x].rig = rig;
    // 區塊有多長，題目詢問的重點
    if(lef == rig){
        tree[x].value = num[lef].fre;
        return;
    }
    int mid = (lef + rig) >> 1;
    build(lef, mid, Lson(x));
    build(mid + 1, rig, Rson(x));
    tree[x].value = max(tree[Lson(x)].value,
        tree[Rson(x)].value);
}
// 查詢 segement tree
int query(int lef, int rig, int x){
    // 題目所查詢的區間剛好在同個區塊上，num[lef].v
        == num[rig].v
    if(num[lef].value == num[rig].value) return rig -
        lef + 1;
    int ans = 0;
    // 查詢的左區間邊界切到區塊，且此區間有數個區塊
    if(lef > num[lef].lef){
        // 計算切到的區間大小
        ans = num[lef].rig - lef + 1;
        //
            更新左邊界至被切區塊的右邊界加一，就不會切到區塊
        lef = num[lef].rig + 1;
    }
    // 查詢的右區間邊界切到區塊，且此區間有數個區塊
    if(rig < num[rig].rig){
        // 計算切到的區間大小，並找出最大
        ans = max(ans, rig - num[rig].lef + 1);
        // 更新右邊界
        rig = num[rig].lef - 1;
    }
    //
        如果左邊界大於右邊界，表示不需要再進行查詢直接回傳答案
    if(lef > rig) return ans;
    if(tree[x].lef >= lef && tree[x].rig <= rig)
        return tree[x].value;
    int mid = (tree[x].lef + tree[x].rig) >> 1;
```

```
    if(lef <= mid) ans = max(ans, query(lef, rig,
        Lson(x)));
    if(mid < rig) ans = max(ans, query(lef, rig,
        Rson(x)));
    return ans;
}
int main(){
    int n, q;
    while(cin >> n && n){
        cin >> q;
        int start = 1;
        for(int i = 1; i <= n; ++i){
            cin >> num[i].value;
            if(num[i].value != num[i-1].value){
                for(int j = start; j < i; ++j){
                    num[j].rig = i - 1;
                    num[j].fre = i - start;
                }
                start = num[i].lef = i;
            }
            else num[i].lef = start;
        }
        // 最後一段 [start, n]
        for(int j = start; j <= n; ++j){
            num[j].rig = n;
            num[j].fre = n - start + 1;
        }
        build(1, n, 1);
        int lef, rig;
        for(int i = 0; i < q; ++i){
            cin >> lef >> rig;
            cout << query(lef, rig, 1) << endl;
        }
    }
}
```

12 Bipartite Graph

12.1 Claw Decomposition

```
/*二分圖 Bipatirate*/
const int maxn = 300+5;
int n;
int color[maxn];
vector<vector<int>> v(maxn);
bool dfs(int s){
    for(auto it : v[s]){
        if(color[it] == -1){
            //
                如果與點相連又還未填色，填塞成與原點不同的
            color[it] = 3 - color[s];
            // 同樣對此點去判定與此點相連的點的填色
            if(!dfs(it)) return false;
        }
        if(color[s] == color[it]){
            // 如果相鄰兩點同色，回傳 false
            return false;
        }
    }
    return true;
}
void isBipatirate(){
    bool flag = true;
    for(int i = 1; i <= n; ++i){
        if(color[i] == -1){
            // 如果還未填色過，就先填色成
                1，並對與此點相連的點都 dfs 判定填色
            color[i] = 1;
            flag &= dfs(i);
        }
    }
    if(flag) cout << "YES" << endl;
    else cout << "NO" << endl;
}
int main(){
    while(cin >> n && n){
        for(int i = 1; i <= n; ++i) v[i].clear();
        memset(color, -1, sizeof(color));
        int a, b;
        while(cin >> a >> b && (a || b)){
            v[a].emplace_back(b);
            v[b].emplace_back(a);
        }
        isBipatirate();
    }
}
```

12.2 Guardian of Decency

```
/* 二分圖最大匹配
匈牙利演算法 Hungarian algorithm*/
const int maxn = 500+5;
int bn, gn;
int match[maxn];
bool visited[maxn];
vector<vector<int>> G(maxn);
struct People{
    int h;
    string music, sport;
    // constructor
    People(){}
    People(int h, string music, string sport){
        this->h = h;
        this->music = music;
        this->sport = sport;
    }
}lef[maxn], rig[maxn];
bool check(People boy, People girl){
    if(abs(boy.h - girl.h) <= 40 && boy.music ==
        girl.music && boy.sport != girl.sport) return
        true;
    return false;
}
bool dfs(int s){
    for(int i = 0; i < G[s].size(); ++i){
        int v = G[s][i];
        if(visited[v]) continue;
        visited[v] = true;
        // 如果這個女生還沒被配對過，直接匹配
        // 如果已經被配對，則根據這個女生所配對的對象
            dfs 重新匹配所有人的對象
        if(match[v] == -1 || dfs(match[v])){
            match[v] = s;
            return true;
        }
    }
    return false;
}
int Hungarian(){
    int cnt = 0;
    memset(match, -1, sizeof(match));
    for(int i = 0; i < bn; ++i){
        memset(visited, false, sizeof(visited));
        if(dfs(i)) cnt++;
    }
    return cnt;
}
int main(){
    int t;
    cin >> t;
    while(t--){
        int N;
        cin >> N;
        bn = 0, gn = 0;
        for(int i = 0; i <= N; ++i) G[i].clear();
        int h;
        string sex, music, sport;
        for(int i = 0; i < N; ++i){
            cin >> h >> sex >> music >> sport;
```

```
            if(sex == "M")
                lef[bn++] = People(h, music, sport);
            else
                rig[gn++] = People(h, music, sport);
        }
        for(int i = 0; i < bn; ++i)
            for(int j = 0; j < gn; ++j)
                if(check(lef[i], rig[j]))
                    G[i].emplace_back(j);
        cout << N - Hungarian() << endl;
    }
}
```

13 Function

13.1 CHAR

```
isdigit()
isalnum() // 判斷字母 || 數字
isalpha()
islower()
isupper()
isblank() // 判斷 即 space 和 \t
toupper()
tolower()
```

13.2 string

```
int main(){
    string str;
    while(cin >> str){
        // substr 取 str idx 2~4 的值
        cout << str.substr(2, 4) << endl;
        // substr 取 str idx 2 以後的所有值
        cout << str.substr(2) << endl;

        string subst;
        cin >> subst;
        // str.append 連接字串
        cout << str.append(subst) << endl;

        char s[100], ss[100];
        cin >> s >> ss;

        char *p;
        // strstr 回傳在s裡找到ss後的整個字串(從 ss
            idx 0 到結束)
        p = strstr(s, ss);
        cout << p << endl;
        // strstr 也可以單純用來找字串
        if(p != NULL) cout << "yes" << endl;
        else cout << "no" << enld;
    }
}
```

13.3 setprecision

```
double cnt = 3.5555;
cout << fixed << setprecision(3) << cnt ;
```

13.4 GCD LCM

```
int gcd(int a, int b){
    return (b == 0 ? a : gcd(b, a % b));
}
int lcm(int a, int b){
    return a * b / gcd(a, b);
}
```

```
}

/* 輾轉相除法 - 求兩數是否互質
如果兩數互質 最終結果其中一方為0時 另一方必為1
若兩數有公因數 最終結果其中一方為0時 另一方必不為1 */
while ( ( num1 %= num2 ) != 0 && ( num2 %= num1 ) !=
    0 );
```

13.5 reverse

```
int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
reverse(a, a + 5);

vector<int> v;
reverse(v.begin(), v.end());

string str = "123";
reverse(str.begin(), str.end());
cout << str << endl; //321
```

13.6 sort

```
priority_queue<int, vector<int>, less<int>> // 大到小
priority_queue<int, vector<int>, greater<int>> //
    小到大

int arr[] = {4, 5, 8, 3, 7, 1, 2, 6, 10, 9};
sort(arr, arr+10);

vector<int> v;
sort(v.begin(), v.end()); //小到大

int cmp(int a, int b){
    return a > b;
}
sort(v.begin(), v.end(), cmp); //大到小
```

13.7 map

```
int main(){
    map<string, string> mp;
    map<string, string>::iterator iter;
    map<string, string>::reverse_iterator iter_r;

    mp.insert(pair<string, string>("r000", "zero"));

    mp["r123"] = "first";

    for(iter = mp.begin(); iter != mp.end(); iter++)
        cout<<iter->first<<" "<<iter->second<<endl;
    for(iter_r = mp.rbegin(); iter_r != mp.rend();
        iter_r++)
        cout<<iter_r->first<<"
            "<<iter_r->second<<endl;

    iter = mp.find("r123");
    mp.erase(iter);

    iter = mp.find("r123");
    if(iter != mp.end())
       cout<<"Find, the value is
           "<<iter->second<<endl;
    else
       cout<<"Do not Find"<<endl;

    mp.clear();
    mp.erase(mp.begin(), mp.end());
}
```

13.8 set

```cpp
int main(){
    set<int> st {1, 6, 8}; // 直接初始化的寫法
    st.insert(1); // 也可以這樣寫就好
    set<int>::iterator iter;

    // 如果有找到，就會傳回正確的 iterator，否則傳回 st.end()
    if (iter != st.end()) {
        cout << "Found: " << *iter << endl;
    } else {
        cout << "Not found." << endl;
    }
    // cout: Found: 6

    // 取值：使用iterator
    x = *st.begin(); // set 中的第一個元素(最小的元素)
    x = *st.rbegin(); // set 中的最後一個元素(最大的元素)

    // search
    iter = st.find(6);
    auto it = st.find(x); // binary search, O(log(N))
    auto it = st.lower_bound(x); // binary search, O(log(N))
    auto it = st.upper_bound(x); // binary search, O(log(N))

    st.clear();
}
```