Contents

1 Data Structure

1.1 Josephus

```
int josephus ( int n , int k) {
    // 有 n 個人圍成一圈，每 k 個一次
    return n > 1 ? ( josephus ( n - 1 , k ) + k ) % n : 0;
}
// 回傳最後一人的編號
, 0 index
```

1.2 BIT

```
#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;
}
```

1.3 Segment tree

```
int dfs(int lef, int rig){
    if(lef + 2 == rig){
        if(num[lef] > num[rig-1]){
            return lef;
        }
        else{
            return rig-1;
        }
    }
    int mid = (lef + rig)/2;
    int p1 = dfs(lef, mid);
    int p2 = dfs(mid, rig);
    if(num[p1] > num[p2]){
        return p1;
    }
    else{
        return p2;
    }
}
```

1.4 Trie

```
const int MAXL = ; // 自己填
const int MAXC = ;
struct Trie {
  int nex[MAXL][MAXC];
  int len[MAXL];
  int sz;
  void init() {
    memset(nex, 0, sizeof(nex));
    memset(len, 0, sizeof(len));
    sz = 0;
  }
  void insert(const string &str) {
    int p = 0;
    for (char c : str) {
      int id = c - 'a';
      if (!nex[p][id]) {
        nex[p][id] = ++sz;
      }
      p = nex[p][id];
    }
    len[p] = str.length();
  }
  vector<int> find(const string &str, int i) {
    int p = 0;
    vector<int> ans;
    for (; i < str.length(); i++) {
      int id = str[i] - 'a';
      if (!nex[p][id]) {
        return ans;
      }
      p = nex[p][id];
      if (len[p]) {
        ans.pb(len[p]);
      }
    }
    return ans;
  }
};
```

2 DP

2.1 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];
}
```

2.2 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();
}
```

3 Graph

3.1 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;
}
```

3.2 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]});
            }
        }
    }
}
```

3.3 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]);
            }
        }
    }
}
```

3.4 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++;
    }
}
```

3.5 KM

```
const int X = 50;   // X的點數，等於Y的點數
const int Y = 50;   // Y的點數
int adj[X][Y];      // 精簡過的adjacency matrix
int lx[X], ly[Y];   // vertex labeling
int mx[X], my[Y];   // X各點的配對對象、Y各點的配對對象
int q[X], *qf, *qb; // BFS queue
int p[X];           // BFS parent，交錯樹之偶點，指向上一個偶點
bool vx[X], vy[Y];  // 記錄是否在交錯樹上
int dy[Y], pdy[Y];  // 表格

void relax(int x){ // relaxation
    for (int y=0; y<Y; ++y)
        if (adj[x][y] != 1e9)
            if (lx[x] + ly[y] - adj[x][y] < dy[y]){
                dy[y] = lx[x] + ly[y] - adj[x][y];
                pdy[y] = x; // 記錄好是從哪個樹葉連出去的
            }
}
void reweight(){ // 調整權重、調整表格
    int d = 1e9;
    for (int y=0; y<Y; ++y) if (!vy[y]) d = min(d, dy[y]);
    for (int x=0; x<X; ++x) if ( vx[x]) lx[x] -= d;
    for (int y=0; y<Y; ++y) if ( vy[y]) ly[y] += d;
    for (int y=0; y<Y; ++y) if (!vy[y]) dy[y] -= d;
}
void augment(int x, int y){ // 擴充路徑
    for (int ty; x != -1; x = p[x], y = ty){
        ty = mx[x]; my[y] = x; mx[x] = y;
    }
}
bool branch1(){ // 延展交錯樹：使用既有的等邊
    while (qf < qb)
        for (int x=*qf++, y=0; y<Y; ++y)
            if (!vy[y] && lx[x] + ly[y] == adj[x][y]){
                vy[y] = true;
                if (my[y] == -1){
                    augment(x, y);
                    return true;
                }
                int z = my[y];
                *qb++ = z; p[z] = x; vx[z] = true; relax(z);
            }
    return false;
}
bool branch2(){ // 延展交錯樹：使用新添的等邊
    for (int y=0; y<Y; ++y){
        if (!vy[y] && dy[y] == 0){
            vy[y] = true;
            if (my[y] == -1){
                augment(pdy[y], y);
                return true;
            }
            int z = my[y];
            *qb++ = z; p[z] = pdy[y]; vx[z] = true; relax(z);
        }
    }
    return false;
}
int Hungarian(){
    // 初始化vertex labeling
    //  memset(lx, 0, sizeof(lx));  // 任意值皆可
    memset(ly, 0, sizeof(ly));
    for (int x=0; x<X; ++x)
        for (int y=0; y<Y; ++y)
            lx[x] = max(lx[x], adj[x][y]);

    // X側每一個點，分別建立等邊交錯樹。
    memset(mx, -1, sizeof(mx));
    memset(my, -1, sizeof(my));
    for (int x=0; x<X; ++x){
        memset(vx, false, sizeof(vx));
        memset(vy, false, sizeof(vy));
        memset(dy, 0x7f, sizeof(dy));
        qf = qb = q;
        *qb++ = x; p[x] = -1; vx[x] = true; relax(x);
        while (true){
            if (branch1()) break;
            reweight();
            if (branch2()) break;
        }
    }
    // 計算最大權完美匹配的權重
    int weight = 0;
    for (int x=0; x<X; ++x)
        weight += adj[x][mx[x]];
    return weight;
}
```

3.6 Dinic

```
// Maximum Flow
const int V = 100, E = 1000;
int adj[V]; // adjacency lists，初始化為-1。
struct Element {int b, r, next;} e[E*2];
int en = 0;
void addedge(int a, int b, int c){
    e[en] = (Element){b, c, adj[a]}; adj[a] = en++;
    e[en] = (Element){a, 0, adj[b]}; adj[b] = en++;
}
int d[V];       // 最短距離
bool visit[V];  // BFS/DFS visit record
int q[V];       // queue
int BFS(int s, int t){ // 計算最短路徑，求出容許圖
    memset(d, 0x7f, sizeof(d));
    memset(visit, false, sizeof(visit));
    int qn = 0;
    d[s] = 0;
    visit[s] = true;
    q[qn++] = s;

    for (int qf=0; qf<qn; ++qf){
        int a = q[qf];
        for (int i = adj[a]; i != -1; i = e[i].next){
            int b = e[i].b;
            if (e[i].r > 0 && !visit[b]){
                d[b] = d[a] + 1;
                visit[b] = true;
                q[qn++] = b;
                if (b == t) return d[t];
            }
        }
    }
    return V;
}
int DFS(int a, int df, int s, int t){ // 求出一條最短擴充路徑，並擴充流量
    if (a == t) return df;
    if (visit[a]) return 0;
    visit[a] = true;
    for (int i = adj[a]; i != -1; i = e[i].next){
        int b = e[i].b;
        if (e[i].r > 0 && d[a] + 1 == d[b]){
            int f = DFS(b, min(df, e[i].r), s, t);
            if (f){
                e[i].r -= f;
                e[i^1].r += f;
                return f;
            }
        }
    }
    return 0;
}
int dinitz(int s, int t){
    int flow = 0;
    while (BFS(s, t) < V)
        while (true){
            memset(visit, false, sizeof(visit));
            int f = DFS(s, 1e9, s, t);
            if (!f) break;
            flow += f;
        }
    return flow;
}
```

4 Other

4.1 Bubble Sort Expect Value

```
/* 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;
    }
}
```

4.2 Crested Ibis vs Monster

```
/* dp 背包 - 重量/價值/可重複使用
因為這題可以重複使用同一條魔法
所以可以這樣dp*/
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;
```

4.3 dpd Knapsack 1

```
// dp 背包 - 時間/數量/價值 - 第幾分鐘符合
int N, W;
cin >> N >> W;
int w[100000+5];
int 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;
```

4.4 Fraction Floor Sum

```
/* [N/i] == M
-> M <= N/i < M + 1
-> N/(M+1) < i <= N/M */
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;
    long long int 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;
```

4.5 Homer Simpson

```
// dp 背包 - 時間/數量 - 漢堡
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);
        }
    }
    if(dp[t] == -1){ // 時間無法剛好吃滿的時候
        for(int i = t; i >= 0; i--){
            if(dp[i] != -1){
                cout << dp[i] << " " << t - i << endl;
                break;
            }
        }
    }
    else{
        cout << dp[t] << endl;
    }
}
```

4.6 Let Me Count The Ways

```
// dp - 時間/數量 - 硬幣排序
long long int n, dp[30000+5];
int coin[] = {1, 5, 10, 25, 50};
memset(dp, 0, sizeof(dp));
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;
    }
}
```

4.7 Luggage

```
// dp 背包 - 重量/是否成立
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;
}
```

4.8 Number of Pairs

```
/* 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 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;
}
```

4.9 ORXOR

```
/* 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 n;
int num[20+7];
memset(num, 0, sizeof(num));
cin >> n;
for(int i = 1; i <= n; i++){
    cin >> num[i];
}
int mini = 2147483647; // 不知道為甚麼只有 2147483647 給過
// 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;
```

4.10 Race to 1

```
const int N = 1000000;
bool sieve[N+5];
vector<int> pri;
double dp[N+5];
void Linear_Sieve(){ // 線性篩
    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]);
    }
    dp[n] = (dp[n] + total)/prime; // 算期望值
    return dp[n];
}
int main(){
    int t;
    int num;
    int 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;
    }
}
```

4.11 SuperSale

```
// dp 背包 - 重量/價值/不可重複使用 - 舉重
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, dp[30+5];
        cin >> pw;
        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;
}
```

4.12 Walking on the Safe Side

```
// dp - 地圖更新
int t;
bool space = false;
cin >> t;
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;
}
```

4.13 X drawing

```
long long int n, a, b, 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;
}
```