Dynamic Programming and Geometric Algorithms: Past Papers Compilation

2025

Contents

1 Matrix Chain Multiplication (DP) 2

2 Graham Scan (Convex Hull) 2

3 KMP Algorithm (Prefix / Failure Function) 3

4 Longest Common Subsequence (LCS) and Shortest Common Supersequence (SCS) 3

5 Counting / Radix Sort for Strings or Player Scores 4

6 Set Union of Two Arrays (O(n) time) 4

7 Closest Pair of Points (Divide Conquer) 4

8 Fibonacci Number (DP, Linear Time) 5

9 Top 100 Player Scores (Radix Sort / Counting Sort) 6

10 Knapsack Problem (DP, Bottom-Up) 7

11 Matrix Chain Multiplication (DP) 8

12 Longest Common Subsequence (LCS) and SCS 9

13 KMP Algorithm (String Matching) 10

14 Graham Scan (Convex Hull) 11

15 Closest Pair of Points (Divide Conquer) 12

16 Union of Two Arrays (O(n) Time) 13

1 Matrix Chain Multiplication (DP)

Problem

Given dimensions of n matrices, find the optimal way to parenthesize them to minimize scalar multiplications.

Recursive Relation

$$M[i,j] = \begin{cases} 0 & i = j \\ \min_{i \le k < j} \{M[i,k] + M[k+1,j] + mat[i-1] * mat[k] * mat[j]\} & i < j \end{cases}$$

Pseudo-code

```
int MatrixChain(int mat[], int i, int j) {
    if (i == j) return 0;
    int minCost = INT_MAX;
    for(int k=i; k<j; k++) {
        int cost = MatrixChain(mat, i, k) + MatrixChain(mat,
           k+1, j) + mat[i-1]*mat[k]*mat[j];
        if(cost < minCost) minCost = cost;
    }
    return minCost;
}
```

2 Graham Scan (Convex Hull)

Problem

Find the convex hull of a set of points in 2D plane.

Algorithm (Pseudo-code)

```
1. Find point P0 with lowest y (and leftmost if tie).
2. Sort all points by polar angle with P0.
3. Initialize empty stack S.
4. Push first 3 points to S.
5. For each remaining point p:
   while (orientation(next-to-top(S), top(S), p) !=
      counterclockwise) pop(S);
   push(S, p);
6. Stack S contains convex hull.
```

3 KMP Algorithm (Prefix / Failure Function)

Problem

Compute prefix function π for pattern string.

Pseudo-code

```
void computePrefix(string pat, int pi[]) {
    int m = pat.length();
    pi[0] = 0;
    int k = 0;
    for(int q=1; q<m; q++) {
        while(k>0 && pat[k]!=pat[q]) k=pi[k-1];
        if(pat[k]==pat[q]) k++;
        pi[q] = k;
    }
}
```

4 Longest Common Subsequence (LCS) and Shortest Common Supersequence (SCS)

Problem

Given strings X and Y, find LCS and then SCS.

Pseudo-code: LCS

```
for i=0 to m:
    for j=0 to n:
        if i==0 or j==0: dp[i][j]=0
        else if X[i-1]==Y[j-1]: dp[i][j]=dp[i-1][j-1]+1
        else: dp[i][j]=max(dp[i-1][j], dp[i][j-1])
```

Pseudo-code: SCS

```
string printSCS(X,Y,dp) {
    i=m; j=n; str="";
    while(i>0 && j>0) {
        if(X[i-1]==Y[j-1]) str.push_back(X[i-1]), i--, j--;
        else if(dp[i-1][j]>dp[i][j-1]) str.push_back(X[i-1]),
            i--;
        else str.push_back(Y[j-1]), j--;
    }
    while(i>0) str.push_back(X[i-1]), i--;
```

```
    while(j>0) str.push_back(Y[j-1]), j--;
    reverse(str.begin(), str.end());
    return str;
}
```

5 Counting / Radix Sort for Strings or Player Scores

Problem

Sort players by score in linear time and get top 100.

Pseudo-code

```
RadixSort(arr):
    max_val = max(arr)
    exp = 1
    while max_val/exp > 0:
        countingSort(arr, exp)
        exp *= 10
```

6 Set Union of Two Arrays (O(n) time)

Problem

Compute union AUB with no duplicates.

Pseudo-code

```
set<int> s;
for(int x: A) s.insert(x);
for(int x: B) s.insert(x);
vector<int> result(s.begin(), s.end());
sort(result.begin(), result.end());
```

7 Closest Pair of Points (Divide Conquer)

Problem

Find the closest pair among n points in 2D plane.

Pseudo-code

```
ClosestPair(points):
    if points.size() <= 3: return bruteForce(points)
    mid = points.size()/2
    dl = ClosestPair(left_half)
    dr = ClosestPair(right_half)
    d = min(dl, dr)
    check strip points within d of mid line
    return min(d, strip_min_distance)
```

8 Fibonacci Number (DP, Linear Time)

Problem: Compute the n-th Fibonacci number in linear time using dynamic programming.

Solution:
Dynamic Programming Principles:

1. Structure of Optimal Solution: $F(n) = F(n-1) + F(n-2)$
2. Principle of Optimality: Solve $F(n-1)$ and $F(n-2)$ first.
3. Bottom-Up Computation: Start from $F(0), F(1)$ and compute iteratively.
4. Optional: Only store last two values to save space.

Pseudocode:

```
function Fibonacci(n):
    if n == 0: return 0
    if n == 1: return 1
    prev2 = 0
    prev1 = 1
    for i = 2 to n:
        current = prev1 + prev2
        prev2 = prev1
        prev1 = current
    return prev1
```

Example Walkthrough for $n = 5$: $F(0) = 0, F(1) = 1$ $i = 2 : F(2) = 0 + 1 = 1$ $i = 3 : F(3) = 1 + 1 = 2$ $i = 4 : F(4) = 1 + 2 = 3$ $i = 5 : F(5) = 2 + 3 = 5$
Complexity: $O(n)$ time, $O(1)$ space.

9 Top 100 Player Scores (Radix Sort / Counting Sort)

Problem: Given $player\_scores = [(ID, score), \dots]$ find top 100 scores efficiently.

Solution:

1. Find maximum score to determine number of digits.
2. Implement counting sort for each digit.
3. Apply radix sort (least to most significant digit).
4. Slice the top 100 scores.

Pseudocode:

```
typedef struct {
    int ID;
    int score;
} Player;

int find_max(Player arr[], int n) {
    int max = arr[0].score;
    for(int i=1;i<n;i++)
        if(arr[i].score > max) max = arr[i].score;
    return max;
}

void counting_sort(Player arr[], int n, int exp) {
    Player output[n];
    int count[10]={0};
    for(int i=0;i<n;i++) count[(arr[i].score/exp)%10]++;
    for(int i=1;i<10;i++) count[i]+=count[i-1];
    for(int i=n-1;i>=0;i--){
        int idx=(arr[i].score/exp)%10;
        output[count[idx]-1]=arr[i]; count[idx]--;
    }
    for(int i=0;i<n;i++) arr[i]=output[i];
}

void radix_sort(Player arr[], int n){
    int max_score = find_max(arr, n);
    for(int exp=1; max_score/exp>0; exp*=10)
        counting_sort(arr,n,exp);
}
```

Complexity: $O(n)$ time with $O(n)$ extra space.

10 Knapsack Problem (DP, Bottom-Up)

Problem: Fit non-essential items into suitcase of remaining weight 8kg to maximize importance.

Items:

Item	Weight	Importance
Book	1	1
Shoes	4	5
Camera	3	4
Clothes	5	7
Toilette	2	3
Laptop	6	9
Passport	7	11

Solution: Standard 0-1 Knapsack DP, bottom-up.

Pseudocode:

```
function knapsack(weights, values, W):
    n = length(weights)
    dp = matrix(n+1, W+1)
    for i = 0 to n:
        for w = 0 to W:
            if i==0 or w==0: dp[i][w] = 0
            else if weights[i-1] <= w:
                dp[i][w] = max(values[i-1]+dp[i-1][w-weights[
                   i-1]], dp[i-1][w])
            else: dp[i][w] = dp[i-1][w]
    return dp[n][W]
```

Optimal Items: Camera (3kg,4) + Clothes (5kg,7) = Total 8kg, Importance 11.

11 Matrix Chain Multiplication (DP)

Problem: Minimize scalar multiplications for chain $A_1 ... A_n$.

Recurrence Relation:

$$M[i,j] = 0 \text{ if } i = j; \quad M[i,j] = \min_{i \le k < j} \{M[i,k] + M[k+1,j] + p_{i-1} p_k p_j\}$$

Recursive Algorithm:

```
function MatrixChain(p, i, j):
    if i==j: return 0
    minCost = INF
    for k in range(i,j):
        cost = MatrixChain(p,i,k) + MatrixChain(p,k+1,j) + p[
           i-1]*p[k]*p[j]
        if cost < minCost: minCost = cost
    return minCost
```

12 Longest Common Subsequence (LCS) and SCS

Problem: X="NOOR", Y="ABDULLAH"

DP Table: Build $dp[m+1][n+1]$ for LCS.

SCS Construction:

```
string printShortestSuperSeq(string X, string Y){
    string str; int i=X.size(), j=Y.size();
    while(i>0 && j>0){
        if(X[i-1]==Y[j-1]) { str.push_back(X[i-1]); i--; j--; }
        else if(dp[i-1][j] > dp[i][j-1]) { str.push_back(Y[j-1]); j--; }
        else { str.push_back(X[i-1]); i--; }
    }
    while(i>0){ str.push_back(X[i-1]); i--; }
    while(j>0){ str.push_back(Y[j-1]); j--; }
    reverse(str.begin(), str.end());
    return str;
}
```

13 KMP Algorithm (String Matching)

Pseudocode:

```
void computeLPSArray(string pat, int M, int lps[]){
    int len = 0, i = 1; lps[0]=0;
    while(i<M){
        if(pat[i]==pat[len]){ len++; lps[i]=len; i++; }
        else{ if(len!=0) len=lps[len-1]; else lps[i]=0; i++;
           }
    }
}

void KMP(string txt, string pat){
    int N=txt.length(), M=pat.length();
    int lps[M];
    computeLPSArray(pat,M,lps);
    int i=0, j=0;
    while(i<N){
        if(pat[j]==txt[i]){ i++; j++; }
        if(j==M){ cout<<"Found at "<<i-j; j=lps[j-1]; }
        else if(i<N && pat[j]!=txt[i]){
            if(j!=0) j=lps[j-1]; else i++;
        }
    }
}
```

14 Graham Scan (Convex Hull)

Pseudocode:

```
function GrahamScan(points):
    p0 = point with lowest y (break ties with lowest x)
    sort points by polar angle with p0
    stack = empty
    push p0, points[0], points[1] to stack
    for i=2 to len(points)-1:
        while len(stack)>1 and orientation(next_to_top(stack)
           , top(stack), points[i])!=CCW:
            pop stack
        push points[i] to stack
    return stack
```

15 Closest Pair of Points (Divide Conquer)

Pseudocode:

```
function closestPair(points):
    sort points by x
    return closestUtil(points)

function closestUtil(Px):
    if len(Px)<=3: return brute_force(Px)
    mid = len(Px)//2
    dl = closestUtil(Px[:mid])
    dr = closestUtil(Px[mid:])
    d = min(dl, dr)
    strip = points within d of mid line
    return min(d, stripClosest(strip, d))
```

16 Union of Two Arrays (O(n) Time)

Problem: Compute $A \cup B$ without duplicates.

Solution (Hash Set / Boolean Array):

```
def union(A,B):
    s = set()
    for x in A: s.add(x)
    for x in B: s.add(x)
    return sorted(list(s))
```

Example: $A = [22, 350, 1, 350, 22, 350, 1]$, $B = [31, 13, 350, 35, 111, 22]$ Union $= [1, 13, 22, 31, 111, 350]$