Projeto de Algoritmos

Programação Dinâmica (2)

João Batista

Programação Dinâmica

- Os exemplos anteriores que vimos são casos não tradicionais de PD
- A seguir veremos alguns casos “clássicos” de PD
  - Range Sum
  - LIS: Longest Increasing Subsequences
  - KnapSack: problema da mochila
  - CC: change Coin
  - TSP (Traveling Salesman Person): ou o famoso caso do Caixeiro Viajante !!!

Range Sum (UVA 507)

- Dada uma sequência de inteiros não zeros, calcule a maior cadeia cuja soma seja máxima.
- RSQ: Range Sum Query...
- Solução TLE (Time Limit Exceeded)
  - Pre-processar o vetor de forma que A[i] += A[i-1]
  - Computar RSQ(i,j)
    - RSQ(0,j) = A[j]
    - RSQ(i,j) = A[j] – A[i-1] para todo i > 0

Range Sum (UVA 507)

- Solução não TLE (Jay Canade)
  - Usar uma variável sum que armazena a soma parcial da cadeia.
    - Sempre que sum < 0, reset o valor...
    - Sum contem: ou o valor acumulado ou representa uma nova “range”...

```
int main() {
  int n = 9, A[] = { 4, -5, 4, -3, 4, 4, -4, 4, -5 };   // a sample array A
  int running_sum = 0, ans = 0;
  for (int i = 0; i < n; i++)                                    // O(n)
    if (running_sum + A[i] >= 0) {  // the overall running sum is still +ve
      running_sum += A[i];
      ans = max(ans, running_sum);          // keep the largest RSQ overall
    }
    else        // the overall running sum is -ve, we greedily restart here
      running_sum = 0;      // because starting from 0 is better for future
                            // iterations than starting from -ve running sum
  printf("Max 1D Range Sum = %d\n", ans);                  // should be 9
} // return 0;
```

LIS: Longest Increasing Subsequence

- N = 8; A = {-7,10,9,2,3,8,8,1}
- Determine a maior cadeia de valores crescentes (não necessariamente continua !
- Os índices associados aos valores sao {0, 1, ..., N-1}
- Neste caso: {-7,2,3,8} !!

LIS

- Seja LIS(i) o valor de LIS que termina no índice i
- Qual a recorrência ?
  - LIS(0) = 1 (se a cadeia termina em -7, só pode ter comprimento 1)
  - LIS(j), j>=1
    - Encontre j (j < i E A[j] < A[i] E LIS(j) é máximo
    - Encontrado j, entao LIS(i) = 1 + LIS(j)
- Mais formalmente
  - LIS(0) = 0
  - LIS(i) = max (1+LIS(j)), para todo j em [0...i-1] !!!

LIS

```
int LIS(int i){
    if (vetLIS[i] != -1)
        return vetLIS[i];
    if (i==0)
        return 1;
    else {
        int ans = INT_MIN;
        for (int j=i-1; j>=0; j--){
            if (A[j] < A[i]) {
                ans = MAX(LIS(j)+1, ans);
            }
        }
        return ans;
    }

}
```

```
int main(){

    //int maior = LIS(n-1);
    memset(vetLIS, -1, sizeof(vetLIS));
    vetLIS[0] = 1;
    int max = 1;
    for (int i=1; i<8; i++){
        vetLIS[i] = LIS(i);
        if (vetLIS[i] > max)
            max = vetLIS[i];
    }


    printf("A maior cadeia tem %d valores\n", max);
```

LIS (versao Halim – mais eficiente)

```
final int MAX_N = 100000;

int n = 11;
int[] A = new int[] {-7, 10, 9, 2, 3, 8, 8, 1, 2, 3, 4};
int[] L_id = new int[MAX_N], P = new int[MAX_N];
Vector<Integer> L = new Vector<Integer>();

int lis = 0, lis_end = 0;
for (int i = 0; i < n; ++i) {
  int pos = Collections.binarySearch(L, A[i]);
  if (pos < 0) pos = -(pos + 1); // some adjustments are needed
  if (pos >= L.size()) L.add(A[i]);
  else                 L.set(pos, A[i]);
  L_id[pos] = i;
  P[i] = pos > 0 ? L_id[pos - 1] : -1;
  if (pos + 1 > lis) {
    lis = pos + 1;
    lis_end = i;
  }

  System.out.printf("Considering element A[%d] = %d\n", i, A[i]);
  System.out.printf("LIS ending at A[%d] is of length %d: ", i, pos + 1);
  reconstruct_print(i, A, P);
  System.out.println("L is now: " + L);
  System.out.printf("\n");
}

System.out.printf("Final LIS is of length %d: ", lis);
```

KnapSack ou o problema da Mochila

- N = 4; V={100,70,50,10} e W={10,4,6,12} e S = 12
- Qual o nro máximo de itens que posso colocar na mochila (pode-se ignorar itens)?
- PD
  - Quem são e quais são os estados?
  - Qual a recorrência?

Mochila

- Estados
  - O(n*S): tem-se n itens e devemos analisá-los para todos os possiveis pesos (S)
- Recorrência: val(it, remW)
  - Val (it, 0) = 0 //encheu a mochila
  - Val(n, remW) = 0; // todos itens considerados!
  - If W[it] > remW → val(it, remW) = val(it+1, remW)
  - If W[it] =< remW
    - val(it, remW) = max(val(it+1,remW), V[it]+val(id+1 - W[it]));

Mochila

```c
int main() {
  scanf("%d", &T);
  while (T--) {
    memset(memo, -1, sizeof memo);

    scanf("%d", &N);
    for (i = 0; i < N; i++)
      scanf("%d %d", &V[i], &W[i]);

    ans = 0;
    scanf("%d", &G);
    while (G--) {
      scanf("%d", &MW);
      ans += value(0, MW);
    }

    printf("%d\n", ans);
  }

  return 0;
}
```

```c
int value(int id, int w) {
  if (id == N || w == 0)
    return 0;
  if (memo[id][w] != -1)
    return memo[id][w];
  if (W[id] > w)
    return memo[id][w] = value(id + 1, w);

  return memo[id][w] = max(value(id + 1, w), V[id] + value(id + 1, w - W[id]));
}
```

Coin Change

- Dado V (quantidade dinheiro) e uma lista de n moedas e seus valores coinValue[i], determine o número mínimo de moedas que representa V
- V = 10; n = 2; coinValue={1,5}
- Estados ??
- Recorrência ??

Coin Change

- Solução greedy, funciona??
  - V = 10; n = 2; coinValue = {1,5}
    - Funciona.. começaria com a moeda de maior valor e chegaria ao resultado: 2 moedas de 5 !!!!!!
  - V = 7, n = 4, coinValue = {1,3,4,5}
    - Nao funcinaria: começaria com moeda = 5 e depois 2 moedas = 1.
    - O correto no entanto: moedas 3 e 4 !!!!!

Coin Change

- Estados
  - Sao apenas os possíveis valores de V, isto é, O(V)
  - Basta verificar os estados para cada moeda
- Recorrência: change (value)
  - Change(0) = 0 (precisamos de 0 moedas para produzir 0 centavos.)
  - Change(< 0) = infinito (inteiro grande...)
  - Change(value) = 1 + min (change(value – coinValue[i])), para i [0..n-1]

Coin Change

```
int change(int value) {
  if (memo[value] != -1)
    return memo[value];

  if (value == 0)
    return 0;
  if (value < 0)
    return INT_MAX;

  int ans = INT_MAX;
  for (int i=0; i<N; i++){
        ans = MIN(1+change(value - coinValue[i]), ans);
    }
  memo[value] = ans;
  return memo[value];
}
```

```
  while (TC--) {
    memset(memo, -1, sizeof memo);

    scanf("%d %d", &N, &V);
    for (int i = 0; i < N; i++)
      scanf("%d", &coinValue[i]);
    printf("V = %d\n", V);
    printf("O numero minimo de moedas eh %d\n", change(V));
```

TSP – Caixeiro Viajante

- Dadas n cidades e a distância entre elas (matriz), calcule o custo mínimo de um caminho de uma cidade s, passando por todas as n-1 cidades, SOMENTE uma vez, retornando a s.
- Problema de permutação
- Alta complexidade para força bruta

TSP e Programação Dinâmica

- Os subproblemas sao repetidos claramente.
- Estados
  - $O(n \times 2^n)$: temos n cidades e temos que lidar com 2^n cidades em cada tour..
  - A complexidade da solucao: cada estado pode ser computado em O(n) portanto $O(2^n \times n^2)$.
- Usar bitmask para representar cada cidade
  - Mask = 18 → 10010 (bit=0 → nao vis; bit=1 → vis)

TSP e Programação Dinâmica

- Recorrência
  - tsp(pos, 2n -1) = dist[pos][0]
    - Todas as cidades visitadas
  - tsp(pos, mask) =

    min(dist[pos][nxt] + tsp(nxt, mask | (1<<nxt)))
    - para todo nxt em [0..n-1]
    - nxt != pos
    - (mask & (1<<nxt)) is '0' (nao visitado!)

TSP

```
int tsp(int pos, int bitmask) {
  // se a todas as cidades ja foram visitadas....
  // retorna a posicao da cidade de origem, fechando o loop...
  if (bitmask == (1 << (n + 1)) - 1)
    return dist[pos][0];
  // evita entradas recursivas desnecessarias, caso ja tenha ocorrido a memoizacao !!!
  if (memo[pos][bitmask] != -1)
    return memo[pos][bitmask];

  // para todas as cidades restantes, executa a comparacao pegando a distancia minima
  int ans = INT_MAX;
  for (int nxt = 0; nxt <= n; nxt++)
    // so entra se a cidade ainda nao foi visitada !!
    if (nxt != pos && !(bitmask & (1 << nxt)))
      ans = min(ans, dist[pos][nxt] + tsp(nxt, bitmask | (1 << nxt)));
  return memo[pos][bitmask] = ans;
}
```

TSP

```
while (TC--) {
  // dimensoes da matriz, que nao precisa ser usadas...
  scanf("%d %d", &xsize, &ysize);
  // guarda a posicao inicial do robo...
  scanf("%d %d", &x[0], &y[0]);

  // le o nro de beeps..
  scanf("%d", &n);
  // guarda as posicoes dos beeps para calculo da distancia..
  for (i = 1; i <= n; i++)
    scanf("%d %d", &x[i], &y[i]);

  // distancia Manhattan para a tabela de distancias...
  for (i = 0; i <= n; i++)
    for (j = 0; j <= n; j++)
      dist[i][j] = abs(x[i] - x[j]) + abs(y[i] - y[j]);

  memset(memo, -1, sizeof memo);
  printf("The shortest path has length %d\n", tsp(0, 1)); // DP-TSP
}
```