Análise dos Problemas do Treino de 23/06/17

G – Cofrinhos da Vó Vitória

- **Problema**

  Dados os depósitos nos cofrinhos de Joãozinho e Zezinho, imprimir a diferença em centavo do valor total do cofrinho.

- **Solução**

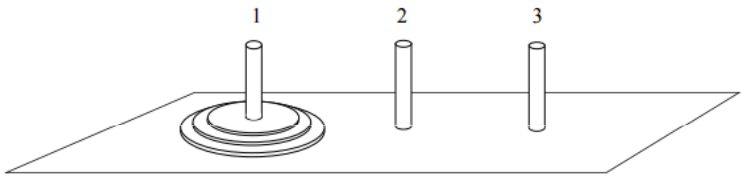
  Apenas implementar o que é pedido.

```
for (i = 0; i < n; i++)
{
    scanf("%d %d", &deposito_joao, &deposito_ze);
    total_joao += deposito_joao;
    total_ze += deposito_ze;
    printf("%d\n", total_joao - total_ze);
}
```

I – Torre de Hanói

- **Problema**

  Dado um algoritmo para resolver a Torre de Hanói e um número N de discos, calcular o número de movimentos que o algoritmo fará.



- **Solução**

  Analisando o algoritmo dado, vemos que quando há 1 disco precisamos de 1 movimento; quando há N discos, precisamos resolver 2 vezes o problema para N-1 discos além de um movimento transferindo o N-ésimo disco. Ou seja:

  $F(1) = 1$

  $F(n) = 1 + 2*F(n-1)$

I – Torre de Hanói

- Podemos calcular F(n) iterativamente:

```
F[1] = 1;
for (i = 2; i <= n; i++)
    F[i] = 1 + 2 * F[i-1];
```

- Outro método é analisar os valores de F(n):

  F(1) = 1

  F(2) = 3

  F(3) = 7

  F(4) = 15

  ....

  F(n) = 2^n - 1

  A fórmula acima pode ser provada usando indução.

H – Estágio

- **Problema**

  Dadas as notas e códigos dos alunos, imprimir o código dos alunos com a maior nota.

- **Solução**

  Implementar o que é pedido.

```
int maior_nota = 0;

for(int i = 0; i < n; i++)
    if(nota[i] > maior_nota)
        maior_nota = nota[i];

printf("Turma %d\n", turma);
for(int i = 0; i < n; i++)
    if(nota[i] == maior_nota)
        printf("%d ", codigo[i]);
```

J - Supermercado

- **Problema**

  Dadas as coordenadas dos quarteirões dos supermercados, calcular o ponto que minimiza a soma da distância de Manhattan entre o centro e entre cada supermercado.

- **Solução**

  Ideia mais simples: força bruta

  Testar todos os pontos possíveis; em cada ponto, calcular a soma das distâncias do ponto para os supermercados; retornar o ponto (um dos) de menor soma.

J - Supermercado

```
int distancia_manhattan(int x0, int y0, int x1, int y1) {
    return abs(x1 - x0) + abs(y1 - y0);
}

for(int x = -1000; x <= 1000; x++)
{
    for(int y = -1000; y <= 1000; y++)
    {
        int dist = 0;
        for(int k = 0; k < n; k++)
            dist += distancia_manhattan(x, y, coord_x[k], coord_y[k]);

        if(dist < melhor_dist)
        {
            melhor_dist = dist;
            melhor_x = x;
            melhor_y = y;
        }
    }
}
```

2000 * 2000 pontos possíveis * 1000 supermercados =

4 * 10^9 operações = TLE

J - Supermercado

- Observação chave: na distância de Manhattan, as coordenadas X e Y são independentes;
- Ao iterarmos por uma linha (Y constante), o valor das somas `abs(coord_y[k] - y)` é constante
- Ao iterarmos por uma coluna (X constante), o valor das somas `abs(coord_x[k] - x)` é constante
- Ou seja, podemos calcular o ponto que minimiza as somas para X e o ponto que minimiza as somas para Y e juntar essas duas coordenadas para obter a resposta
- Transformamos o nosso problema em 1D: dado um conjunto de valores, encontrar o valor X que minimiza a distância absoluta entre X e os demais valores

J - Supermercado

- Força bruta agora funciona:

  2000 pontos possíveis * 1000 supermercados para X

  + 2000 pontos possíveis * 1000 supermercados para Y

  = 4 * 10^6 = ACC

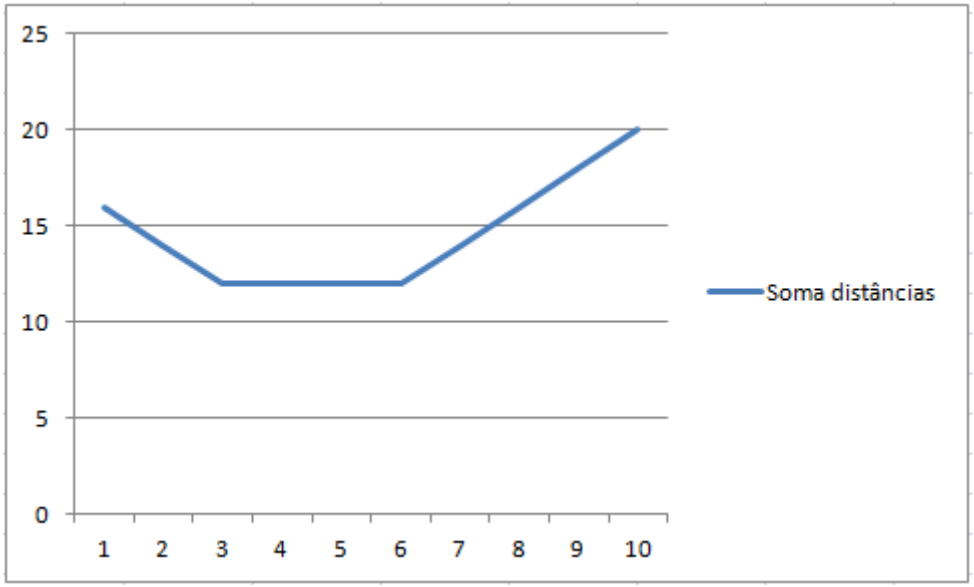
```
for(int x = -1000; x <= 1000; x++)
{
    int dist = 0;
    for(int k = 0; k < n; k++)
        dist += abs(x – coord_x[k]);

    if(dist < melhor_dist)
    {
        melhor_dist = dist;
        melhor = x;
    }
}
```

E a mesma coisa para Y.

J - Supermercado

- Analisando um pouco mais, podemos perceber a seguinte propriedade: o valor de ‘dist’ sempre diminui conforme nos aproximamos do(s) ponto(s) ótimo(s). Ao passar do ponto ótimo, o valor de ‘dist’ sempre aumenta.
- Exemplo: para 4 pontos: 1, 3, 6, 10
  - Ponto ótimo entre 3 e 6

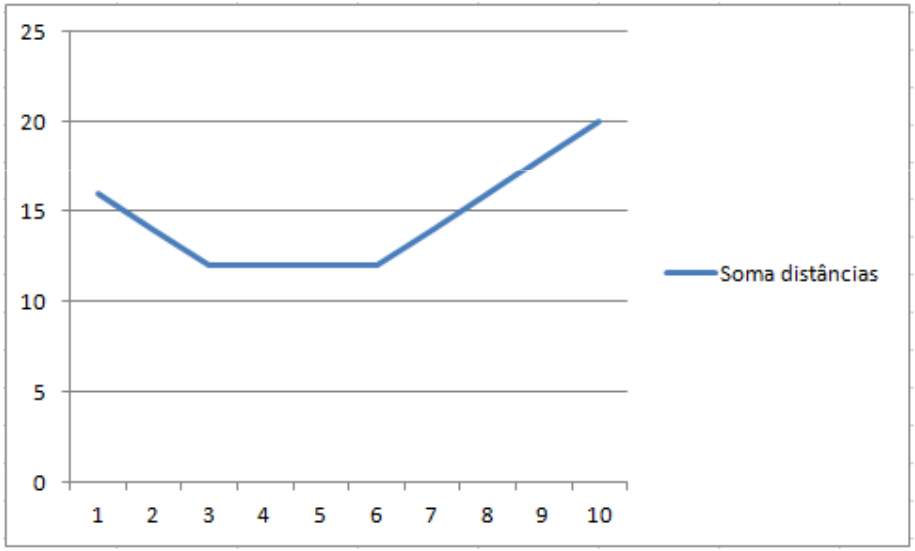


J - Supermercado

- Então podemos fazer uma busca mais inteligente
- Por exemplo, se testarmos os pontos 0, 5 e 10, e observarmos que f(0) > f(5) > f(10), então já sabemos que o ponto ótimo não está entre 0 e 5, pois ou f(5) e f(10) estão na parte decrescente (nesse caso o ponto ótimo é >= 10) ou f(5) está na parte decrescente e f(10) na crescente (nesse caso o ponto ótimo está entre 5 e 10).
  - Essa é a ideia base da **busca ternária**, que serve para determinar o menor ou maior ponto de uma função unimodal (uma função que é estritamente decrescente, depois atinge 1 único valor mínimo, depois estritamente crescente); mas neste caso a busca ternária não funciona, pois a função da soma das distâncias pode possuir vários valores mínimos

J - Supermercado

- Ao invés da busca ternária, podemos utilizar uma busca binária para localizar o 1º ponto tal que f(x) > f(x-1). Aí nossa resposta será o ponto anterior a este ponto.



P(x) = F F F F F F V V V V

- 2 * log2 (2000) pontos testados * 1000 supermercados =
  ~ 22.000 operações

J - Supermercado

- Com um pouco mais de análise, podemos perceber que o ponto ótimo será sempre a mediana do conjunto (se N for ímpar) ou qualquer ponto entre os dois pontos que determinam a mediana (se N for par).
- Intuitivamente: consideremos o custo no ponto mais à esquerda de todos. Chamemos esse custo de C. Ao avançar 1 para a direita, nosso novo custo passa a ser

  $C' = C + 1 - (N-1)$

  Pois estamos nos afastando de 1 ponto e nos aproximando de N-1 pontos. Ao avançar mais 1 para a direita, teremos

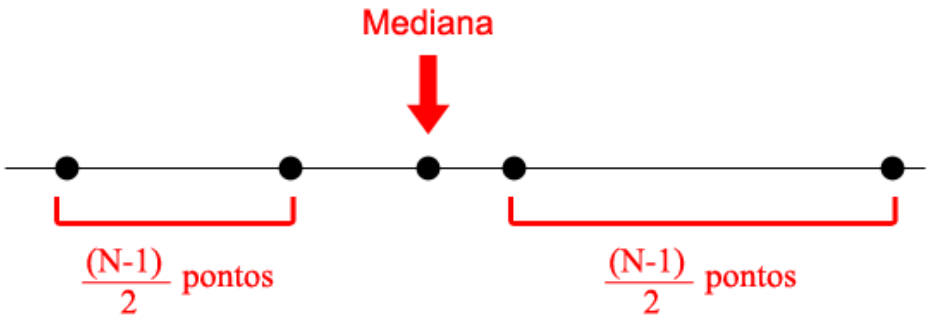
  $C'' = C' + 1 - (N-1)$

  e assim por diante até encontrar o segundo ponto mais à esquerda, onde passaremos a ter o seguinte novo custo:

  $C''' = C'' + 2 - (N-2)$

J - Supermercado

- Se continuarmos avançando a esquerda, obteremos sempre um novo custo menor do que o custo anterior, até o momento em que começaremos a nos afastar de mais pontos do que estamos nos aproximando, isto é, a partir do momento em que tivermos mais pontos à esquerda do ponto testado do que à direita do ponto testado. Isto acontecerá exatamente após a mediana.



J - Supermercado

- Outro modo de ver:
- Se N for ímpar, na mediana temos N/2 elementos para a esquerda e N/2 elementos para a direita; ao deslocarmos o ponto testado 1 para a direita, nos aproximamos de N/2 elementos e nos afastamos de N/2+1 elementos. Ao deslocarmos para a esquerda, mesma coisa, logo para ambos os lados o custo cresce, logo o custo mínimo é na mediana. Podemos aplicar um argumento similar para o caso em que N é par.

```
Ler os pontos para um vetor X e vetor Y
Ordenar X
Ordenar Y
Resposta = ponto (X[n/2], Y[n/2])
```

E - Penta

- **Problema**

  Dados os jogadores que devem estar no palanque em cada quadra e a capacidade do palanque, imprimir o menor número de trocas possível para satisfazer a restrição.

- **Solução**

  Algoritmo guloso

  Duas etapas: na primeira etapa, o palanque começa vazio; preenchemos o palanque até que este fique cheio

  A partir daí, toda vez que quisermos adicionar um jogador que não está no palanque precisamos remover um jogador; que jogador remover?

E - Penta

- Solução gulosa: escolher o que parece melhor no momento
- No caso, escolhemos para descer do palanque o jogador que for levar mais tempo para voltar a subir ao palanque (ou que não voltará mais)
- Vamos tentar provar que essa ideia leva a solução ótima:

E - Penta

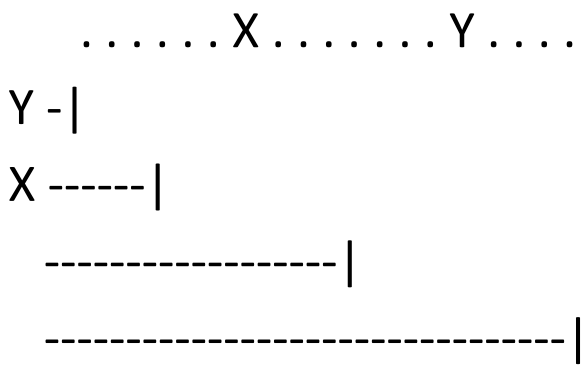
- Vamos supor que o jogador que está no palanque e que levará mais tempo para voltar é o jogador Y. Vamos supor que nossa solução não remove Y, mas sim remove X (que voltará ao palanque antes de Y).

```
      . . . . . . X . . . . . . . Y . . . .
 X -|
 Y ------|
   -----------------|
   -----------------------------|
```

- Temos 3 casos possíveis:
  - Y desce antes de X reaparecer (+1 para X e +1 para Y = 2 trocas)
  - Y desce entre o reaparecimento de X e o reaparecimento de Y (+1 para X e +1 para Y = 2 trocas)
  - Y desce apenas após seu reaparecimento no palanque (+1 para X = 1 troca)

E - Penta

- Se, ao invés de descermos Y, descermos X, teremos a seguinte opção:



- Temos 3 casos possíveis:
  - X desce antes de X reaparecer (+1 para X e +1 para Y = 2 trocas)
  - X desce entre o reaparecimento de X e o reaparecimento de Y (+0 para X e +1 para Y = 1 troca)
  - X desce apenas após o reaparecimento de Y no palanque (+1 para Y = 1 troca)

E - Penta

- Nos três casos, escolher a descida de Y nos leva a uma solução melhor ou igual a solução obtida descendo X:

	Descer X	Descer Y
Caso 1	2	2
Caso 2	2	1
Caso 3	1	1

- Logo, descer Y nunca será pior que descer X. Portanto a solução ótima pode ser conseguida com essa estratégia.

E - Penta

- Como calcular a próxima vez que o jogador aparecerá:
- São 10.000 quadras, então se toda vez que um novo jogador for subir ao palanque, calcularmos a próxima aparição para os jogadores que já estão no palanque (até 22 jogadores), teríamos aprox. 10.000 quadras * 22 jogadores no palanque * 10.000 quadras para procurar (no pior caso a próxima aparição vai estar lá no fim) = 2,2 * 10^9 operações = TLE
- Se fizermos desse jeito, estamos recalculando a mesma coisa toda vez
- Duas abordagens:
  - Armazenar para cada jogador a próxima aparição; recalcular apenas para o jogador que está subindo no palanque
  - Criar 23 vetores (matriz 23 x 10.000) e armazenar em cada um as aparições de cada jogador; nesse caso, precisamos também armazenar, para cada vetor, o índice do elemento em que estamos

Demais problemas

- A serem discutidos futuramente:

  Dengue, Pedágio, Passeio – Grafos

  Biblioteca Ótima – Programação Dinâmica

  Dominó - Backtracking