Programação Dinâmica Aplicada a Competições de Programação

MEDITEC 6

7 de Maio de 2015

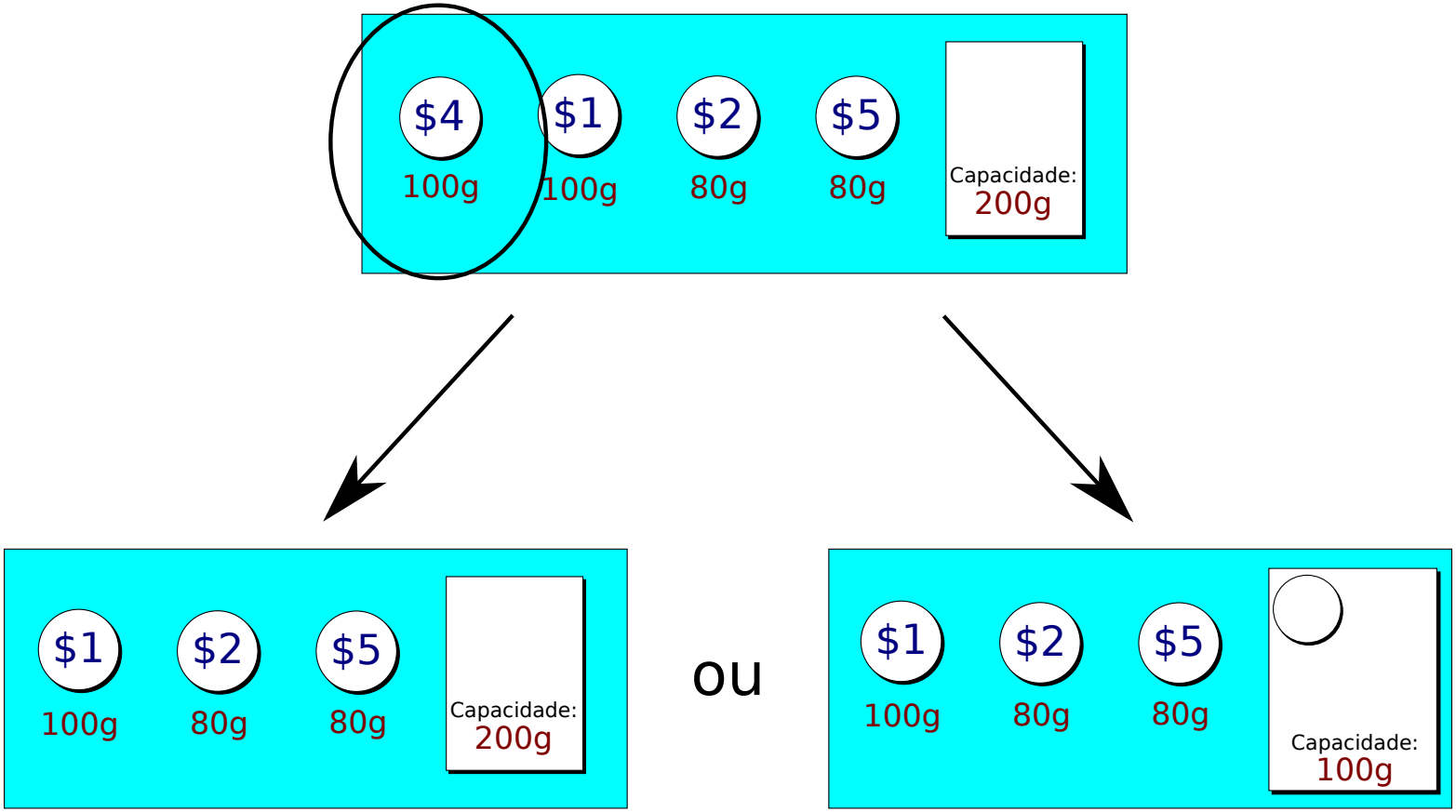
Introdução

- **P**rogramação **D**inâmica
  - Técnica de construção de *algoritmos*
  - Tema **muito frequente** em competições de programação
    - *Must-Know*!

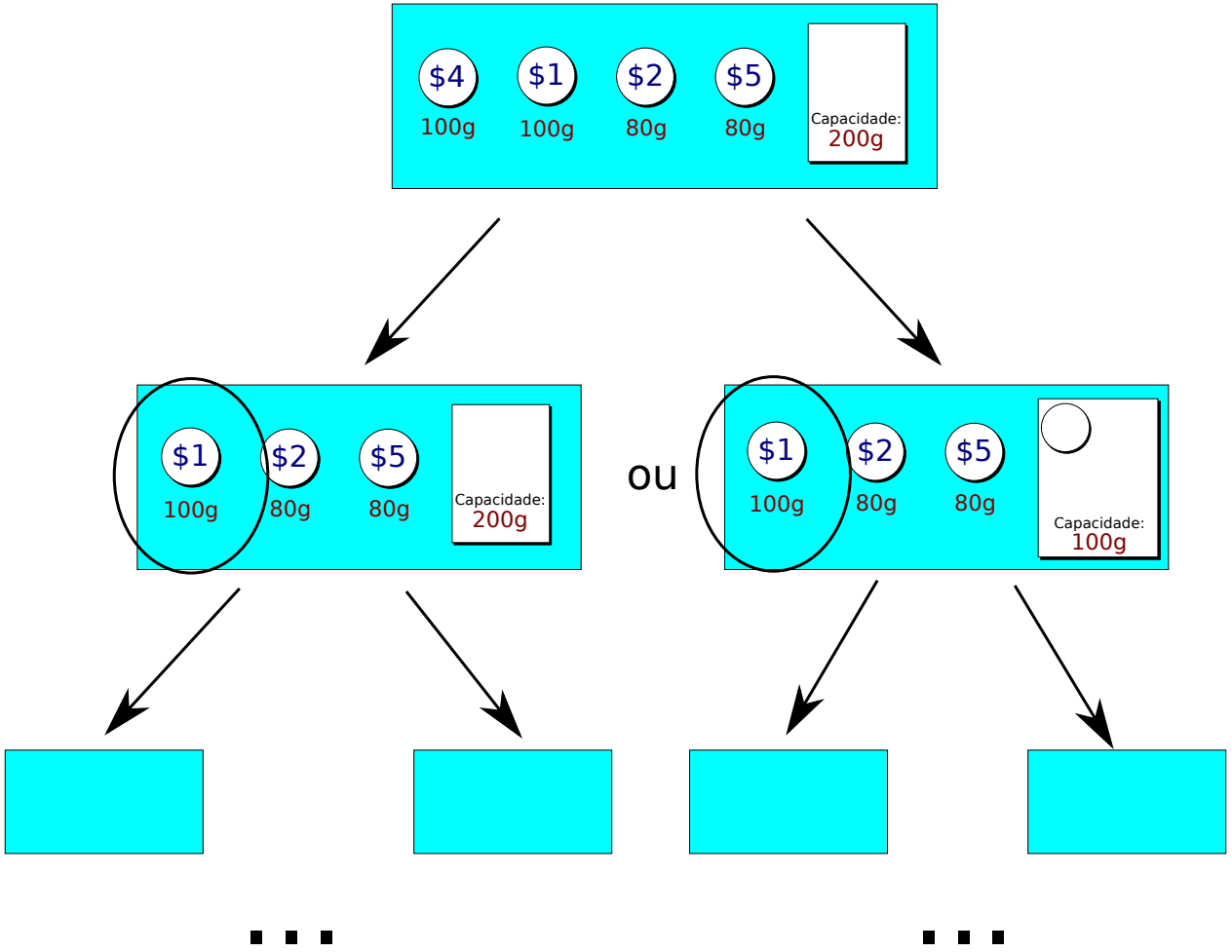
O Problema da Mochila

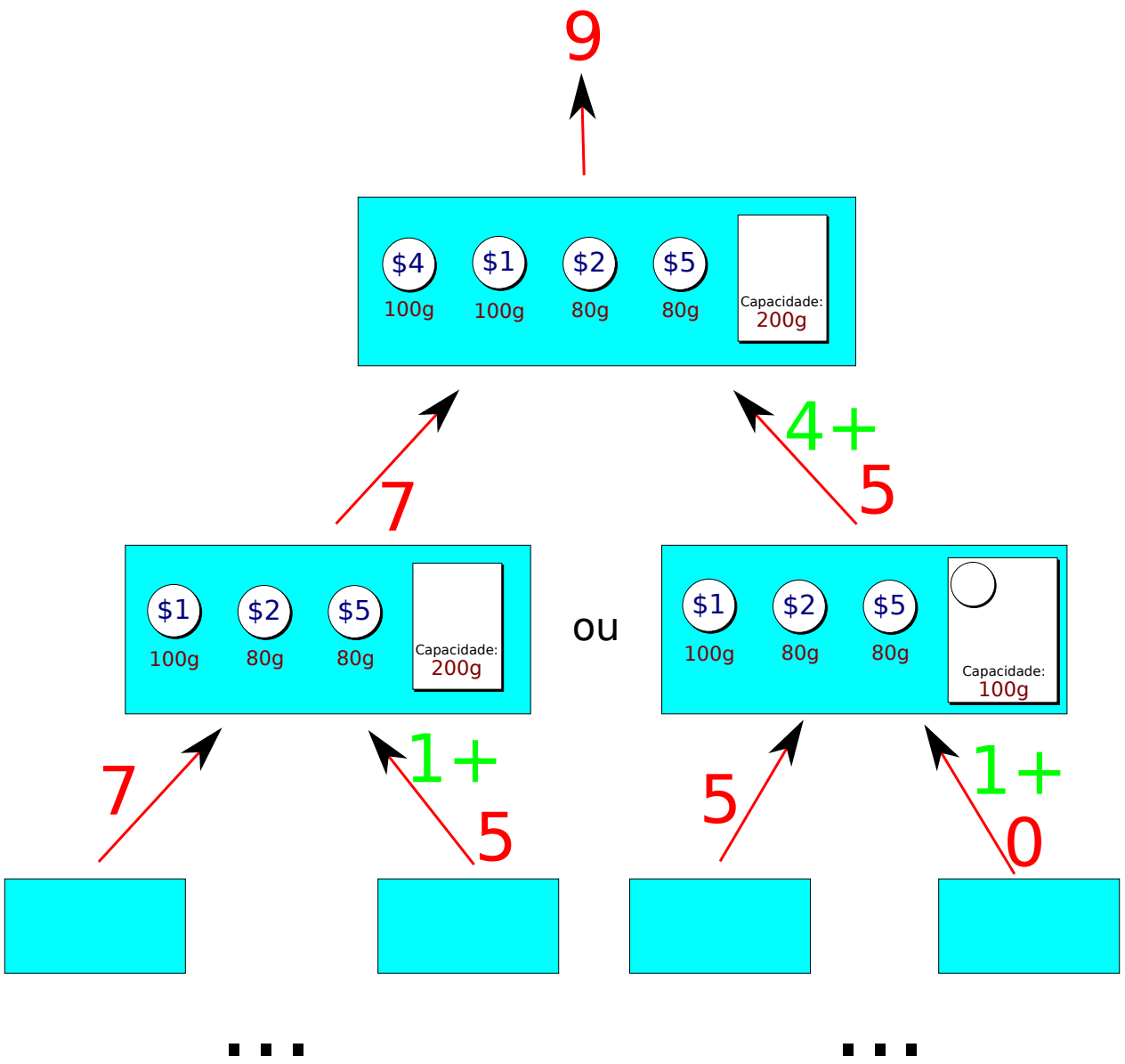
- Maximizar valor total na mochila, respeitando sua capacidade

Força Bruta


$4
100g
$1
100g
$2
80g
$5
80g
Capacidade:
200g
$1
100g
$2
80g
$5
80g
Capacidade:
200g
ou
$1
100g
$2
80g
$5
80g
Capacidade:
100g

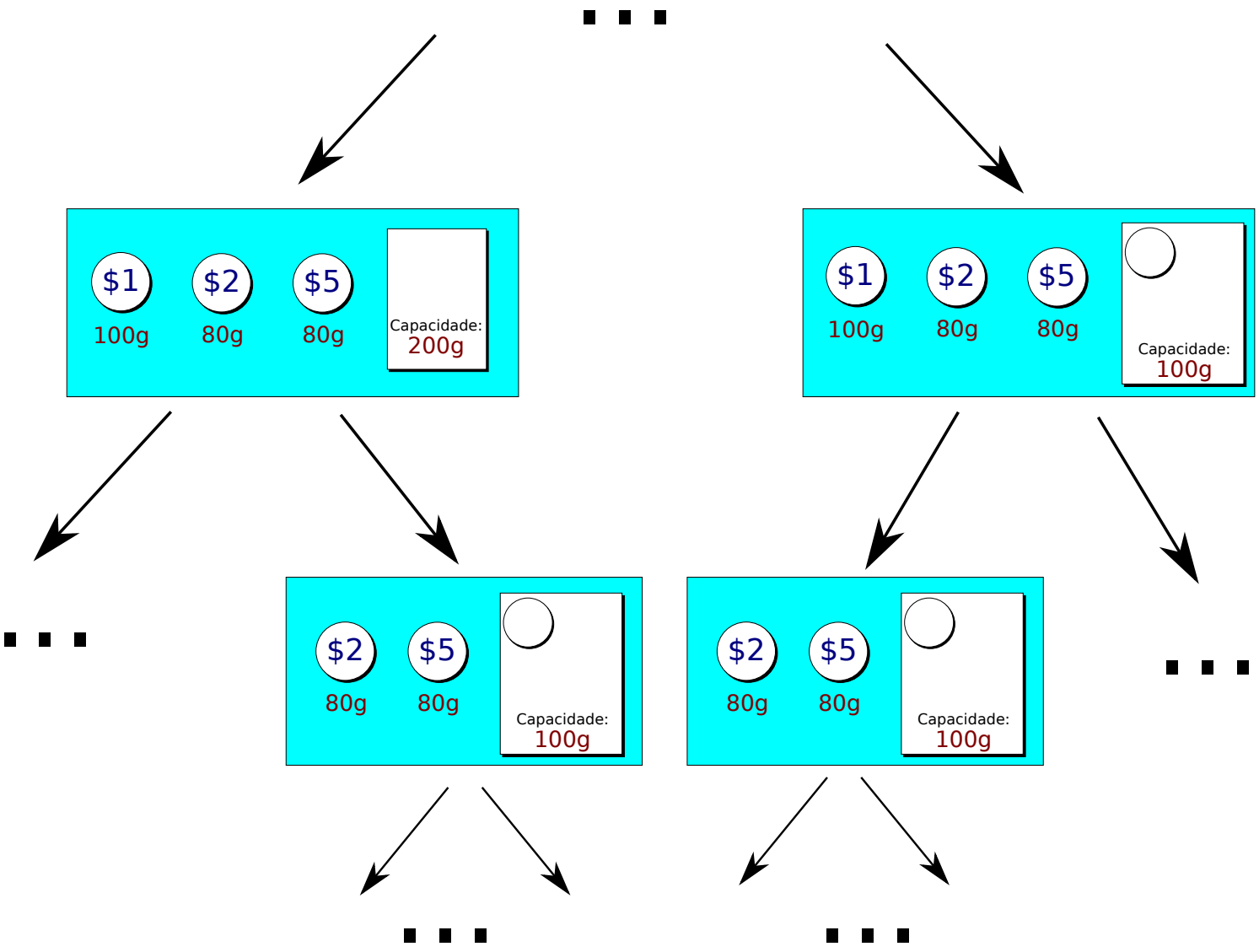
Recursão


$4
100g
$1
100g
$2
80g
$5
80g
Capacidade:
200g
$1
100g
$2
80g
$5
80g
Capacidade:
200g
ou
$1
100g
$2
80g
$5
80g
Capacidade:
100g
...
...


9
$4
100g
$1
100g
$2
80g
$5
80g
Capacidade:
200g
7
4+
5
$1
100g
$2
80g
$5
80g
Capacidade:
200g
ou
$1
100g
$2
80g
$5
80g
Capacidade:
100g
7
1+
5
5
1+
0
...
...

Rascunho de solução

```
int mochila(int item, int capacidade) {
  ...
  // Sem o item atual
  int opc1 = mochila(item+1, capacidade);
  ...
  // Com o item atual
  int opc2 = valor[item] +
       mochila(item+1, capacidade-peso[item]);
  ...
  return max(opc1, opc2);
}
```


...
$1
100g
$2
80g
$5
80g
Capacidade:
200g
$1
100g
$2
80g
$5
80g
Capacidade:
100g
...
$2
80g
$5
80g
Capacidade:
100g
$2
80g
$5
80g
Capacidade:
100g
...
...
...

Those who cannot remember the past are condemned to repeat it.

-Dynamic Programming

Memorização

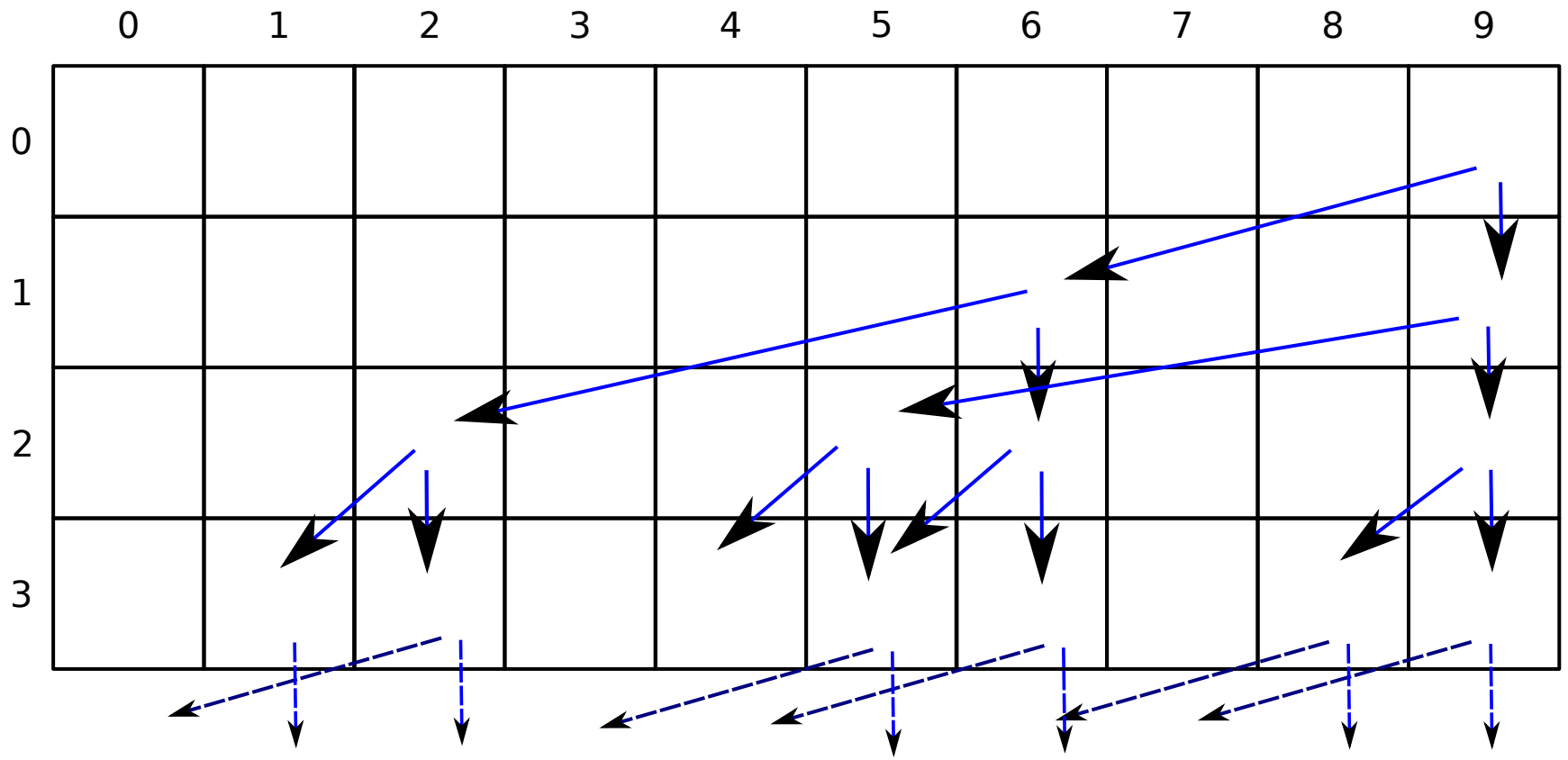
```
int mem[1024][1024]; // Matriz global
```

- Uma convensão:
  - mem[item][capacidade] = -1 se mochila(item,capacidade) ainda nao calculado
  - mem[item][capacidade] = sua resposta caso contrario

Rascunho de solução

```
int mochila(int item, int capacidade) {
if (mem[item][capacidade] != -1)
    return mem[item][capacidade];
  ...
  // Sem o item atual
  int opc1 = mochila(item+1, capacidade);
  ...
  // Com o item atual
  int opc2 = valor[item] +
      mochila(item+1, capacidade-peso[item]);
  ...
  mem[item][capacidade] = max(opc1,opc2);
  return max(opc1, opc2);
}
```

Dependências na matriz



Situação final da matriz

	0	1	2	3	4	5	6	7	8	9
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	12
1	-1	-1	-1	-1	-1	-1	8	-1	-1	10
2	-1	-1	5	-1	-1	7	7	-1	-1	7
3	-1	0	2	-1	2	2	2	-1	2	2

Recorrência

$$mochila(N, cap) = 0$$

$$mochila(item, cap) = \max \begin{cases} mochila(item + 1, cap) \\ valor_{item} + mochila(item + 1, cap - peso_{item}) \\ \qquad \text{se } peso_{item} \leq cap, 0 \text{ c.c.} \end{cases}$$

Custo Computacional

- Espaço
  - Tamanho da matriz
- Tempo
  - Considerar tamanho da matriz
  - Cada posição da matriz é visitada *no máximo* uma vez
  - Considerar custo para visitar cada posição

Custo Computacional

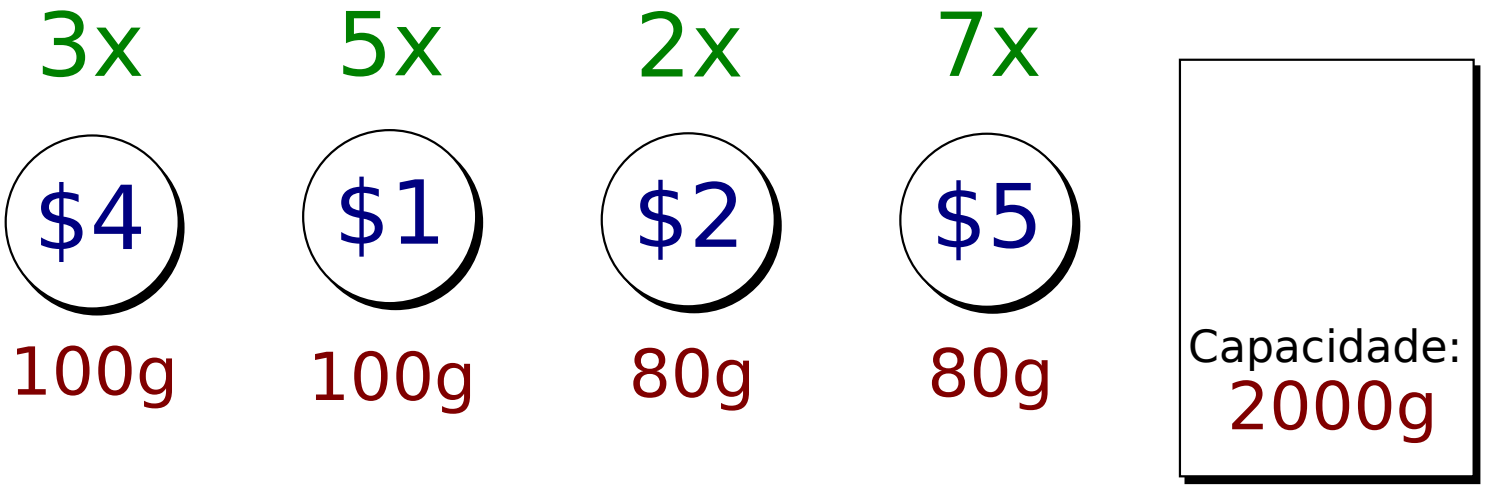
N C

$$mochila(item, cap) = \max \begin{cases} mochila(item+1, cap) \\ valor_{item} + mochila(item+1, cap - peso_{item}) \\ \qquad \text{se } peso_{item} \leq cap, 0 \text{ c.c.} \end{cases}$$

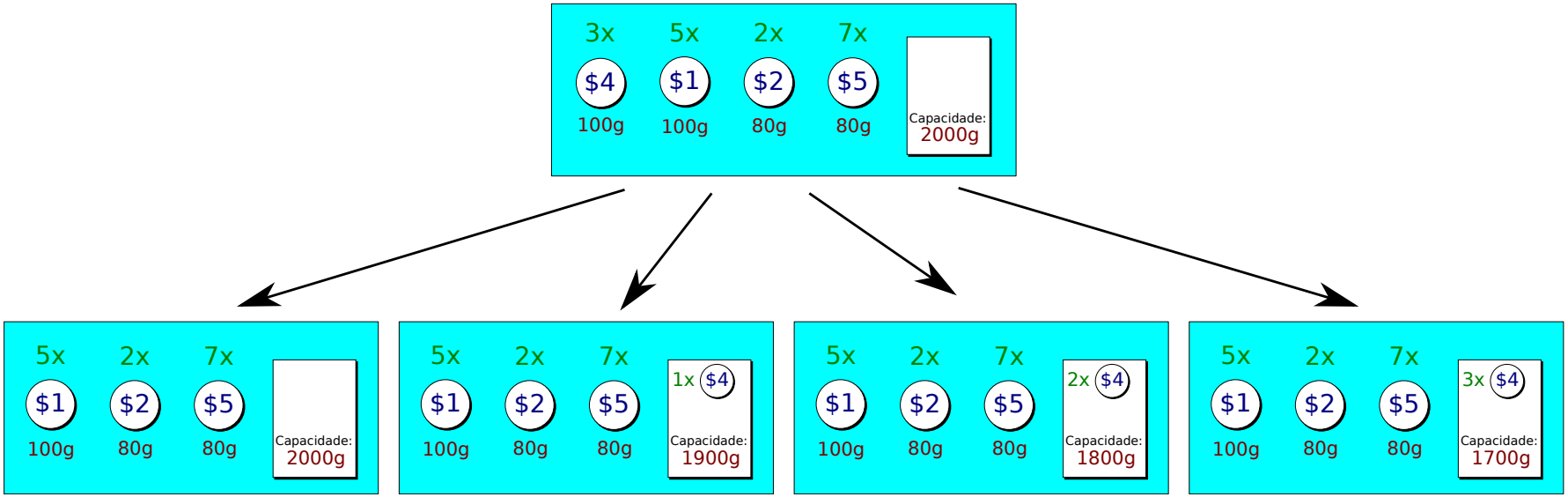
O(1)*

- Espaço
  - O(estados) $=$ O($N \times C$)
- Tempo
  - O(estados) $\times$ O(cada estado)
  - O($N \times C$) $\times$ O(1) $=$ **O($N \times C$)**

O Problema da Mochila (Multi-item)



Recursão


3x 5x 2x 7x
$4 $1 $2 $5
100g 100g 80g 80g
Capacidade: 2000g
5x 2x 7x
$1 $2 $5
100g 80g 80g
Capacidade: 2000g
5x 2x 7x
$1 $2 $5
100g 80g 80g
1x $4
Capacidade: 1900g
5x 2x 7x
$1 $2 $5
100g 80g 80g
2x $4
Capacidade: 1800g
5x 2x 7x
$1 $2 $5
100g 80g 80g
3x $4
Capacidade: 1700g

Rascunho de solução

```
int mochila(int item, int capacidade) {
if (mem[item][capacidade] != -1)
    return mem[item][capacidade];
  ...
  int maior = 0;
  for (int qts=0;qts<=qtde[items];qts++) {
      int opc = qts*valor[item] +
          mochila(item+1, capacidade-qts*peso[item]);
      maior = max(maior, opc);
  }
  mem[item][capacidade] = maior;
  return maior;
}
```

Recorrência

$$mochila(N, cap) = 0$$

$$mochila(item, cap) = \max_{0 \leq q \leq qtde_{item}} \begin{cases} q \times valor_{item} + mochila(item + 1, cap - q \times peso_{item}) \\ \quad \text{se } q \times peso_{item} \leq cap, 0 \text{ c.c.} \end{cases}$$

Custo Computacional

N C

$$mochila(item, cap) = \max_{0 \leq q \leq qtde_{item}} \begin{cases} q \times valor_{item} + mochila(item+1, cap - q \times peso_{item}) \\ \text{se } q \times peso_{item} \leq cap, 0 \text{ c.c.} \end{cases}$$

$O(\max\{qtde_{item}\})$

- Espaço
  - $O(\text{estados}) = O(N \times C)$
- Tempo
  - $O(\text{estados}) \times O(\text{cada estado})$
  - $O(N \times C) \times O(max\{qtde_{item}\}) = \mathbf{O(}N \times C \times max\{qtde_{item}\}\mathbf{)}$

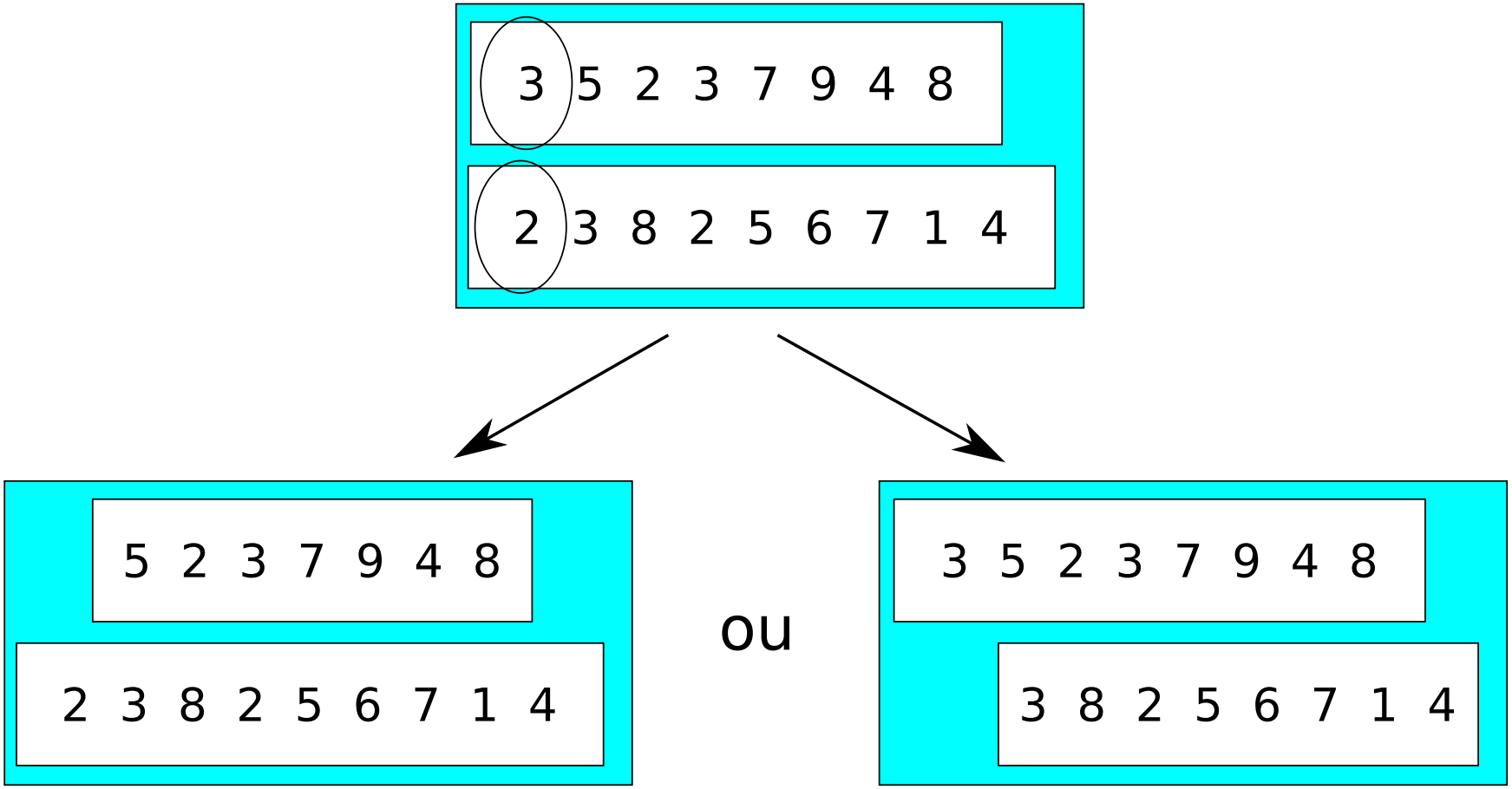
Problema da maior subsequencia comum (LCS)

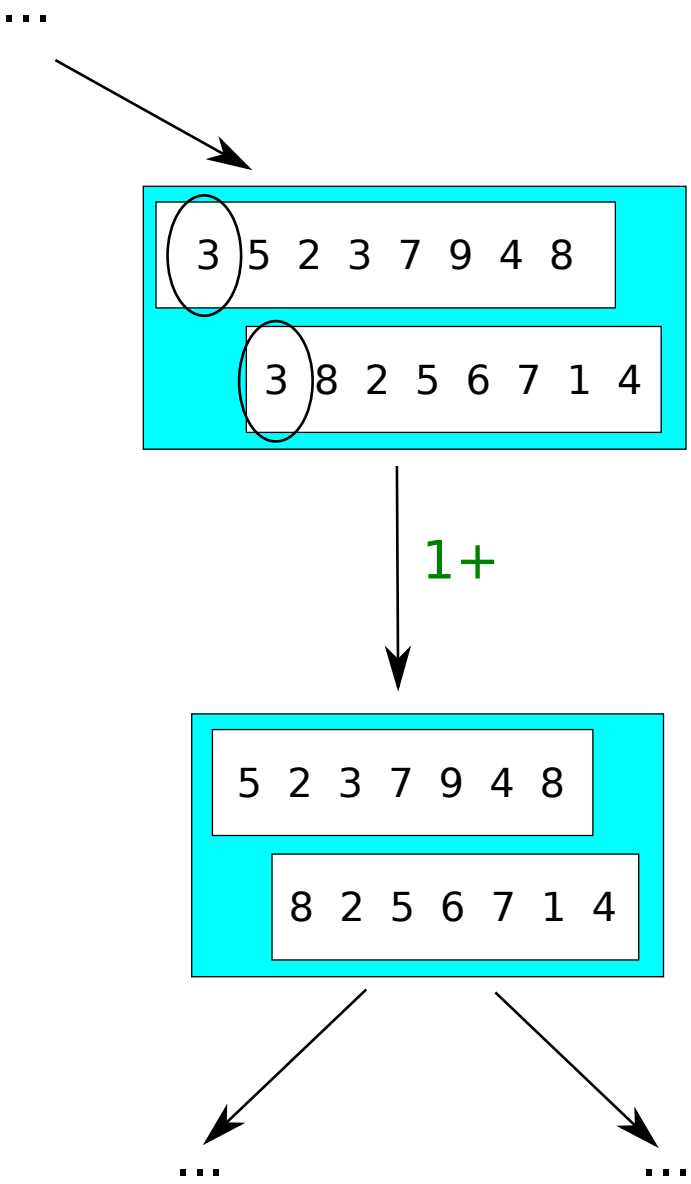
- Dados dois vetores, encontrar *o tamanho* da maior subsequência comum a duas
- `3 5 2 3 7 9 4 8`
- `2 3 8 2 5 6 7 1 4`

Problema da maior subsequencia comum (LCS)

- 3 5 2 3 7 9 4 8
- 2 3 8 2 5 6 7 1 4
- Resposta: 4

Recursão


3 5 2 3 7 9 4 8
2 3 8 2 5 6 7 1 4
5 2 3 7 9 4 8
2 3 8 2 5 6 7 1 4
ou
3 5 2 3 7 9 4 8
3 8 2 5 6 7 1 4


...
3 5 2 3 7 9 4 8
3 8 2 5 6 7 1 4
1+
5 2 3 7 9 4 8
8 2 5 6 7 1 4
...
...

Recorrência

$$LCS(i,j) = \begin{cases} 1 + LCS(i+1, j+1) \text{ se } a_i = b_j \\ max\{LCS(i+1,j), LCS(i,j+1)\} \text{ c.c.} \end{cases}$$

$$LCS(N,j) = LCS(i,M) = 0$$

Custo Computacional

N M

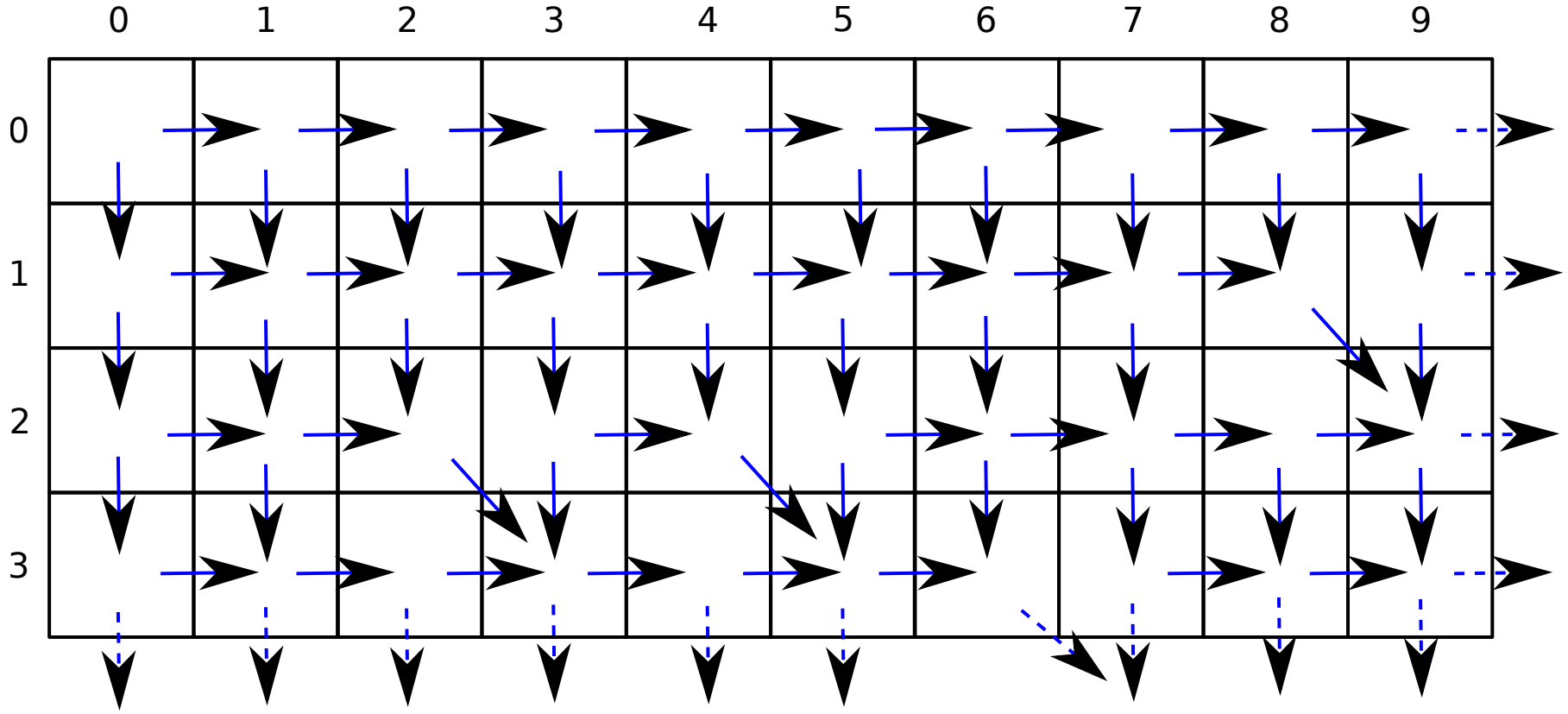
$$LCS(i,j) = \begin{cases} 1 + LCS(i+1, j+1) \text{ se } a_i = b_j \\ max\{LCS(i+1,j), LCS(i,j+1)\} \text{ c.c.} \end{cases}$$

O(1)

- Espaço
  - O(estados) $=$ O($N \times M$)
- Tempo
  - O(estados) $\times$ O(cada estado)
  - O($N \times M$) $\times$ O(1) $=$ **O(**$N \times M$**)**

Dependências na matriz

- 6 3 9 2
- 4 7 9 8 9 8 2 1 3 5



Situação final da matriz

	0	1	2	3	4	5	6	7	8	9
0	2	2	2	2	2	1	1	1	1	0
1	2	2	2	2	2	1	1	1	1	0
2	2	2	2	2	2	1	1	0	0	0
3	1	1	1	1	1	1	1	0	0	0

Rascunho de solução **iterativa**

```
...
for (int i=...)
  for (int j=...)
    if (a[i]==b[j])
      PD[i][j] = 1 + PD[i+1][j+1];
    else
      PD[i][j] = max(PD[i+1][j], PD[i][j+1]);
```

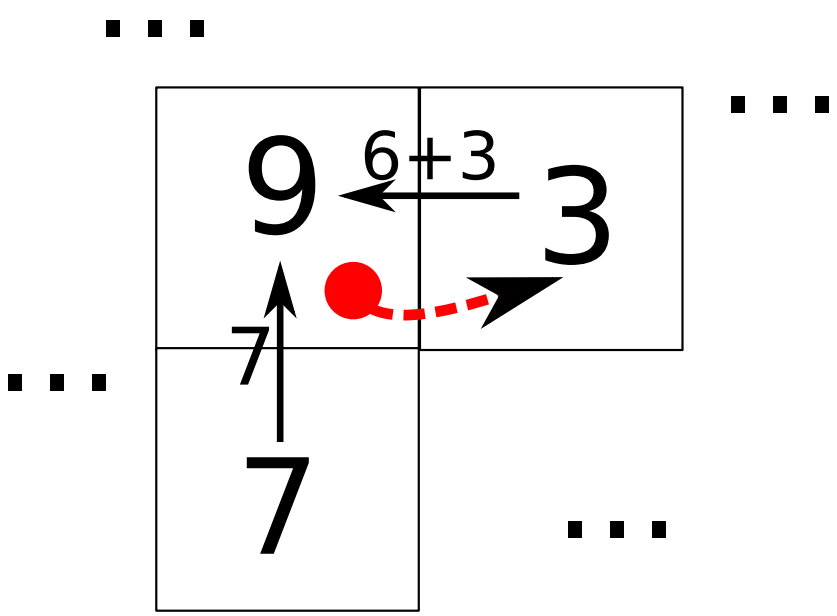
PD Iterativa $\times$ Recursiva

- PD iterativa é mais rápida se *todos* os estados são visitados
  - Necessário formular a *ordem* de visita
- PD recursiva pode ser mais rápida se *nem todos* os estados são visitados
  - Ordem dada "automaticamente" pela recursão

Explicitando a resposta

```
Entrada:

8 9
3 5 2 3 7 9 4 8
2 3 8 2 5 6 7 1 4

Saida

4
3 5 7 4
```

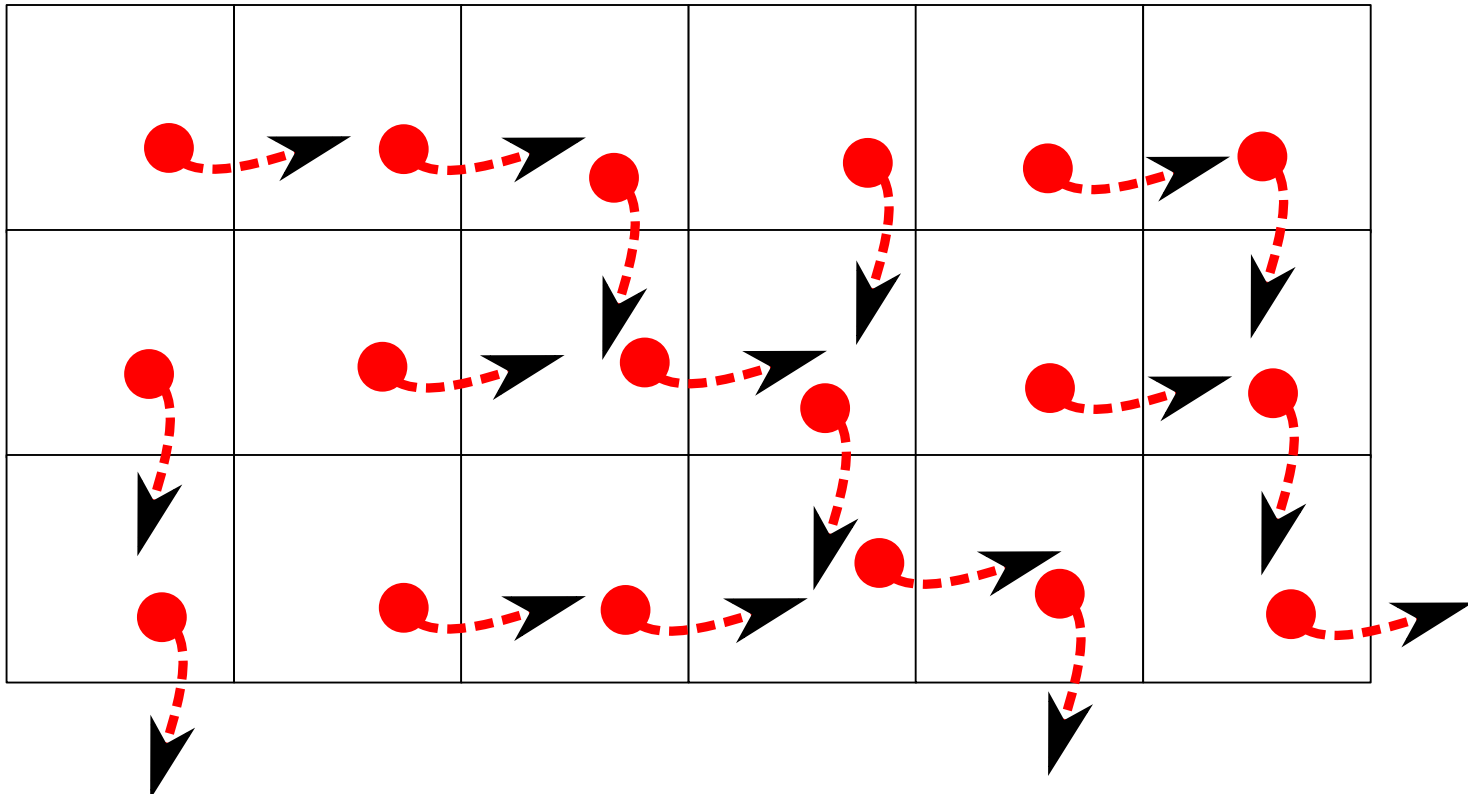
Memorize a melhor escolha



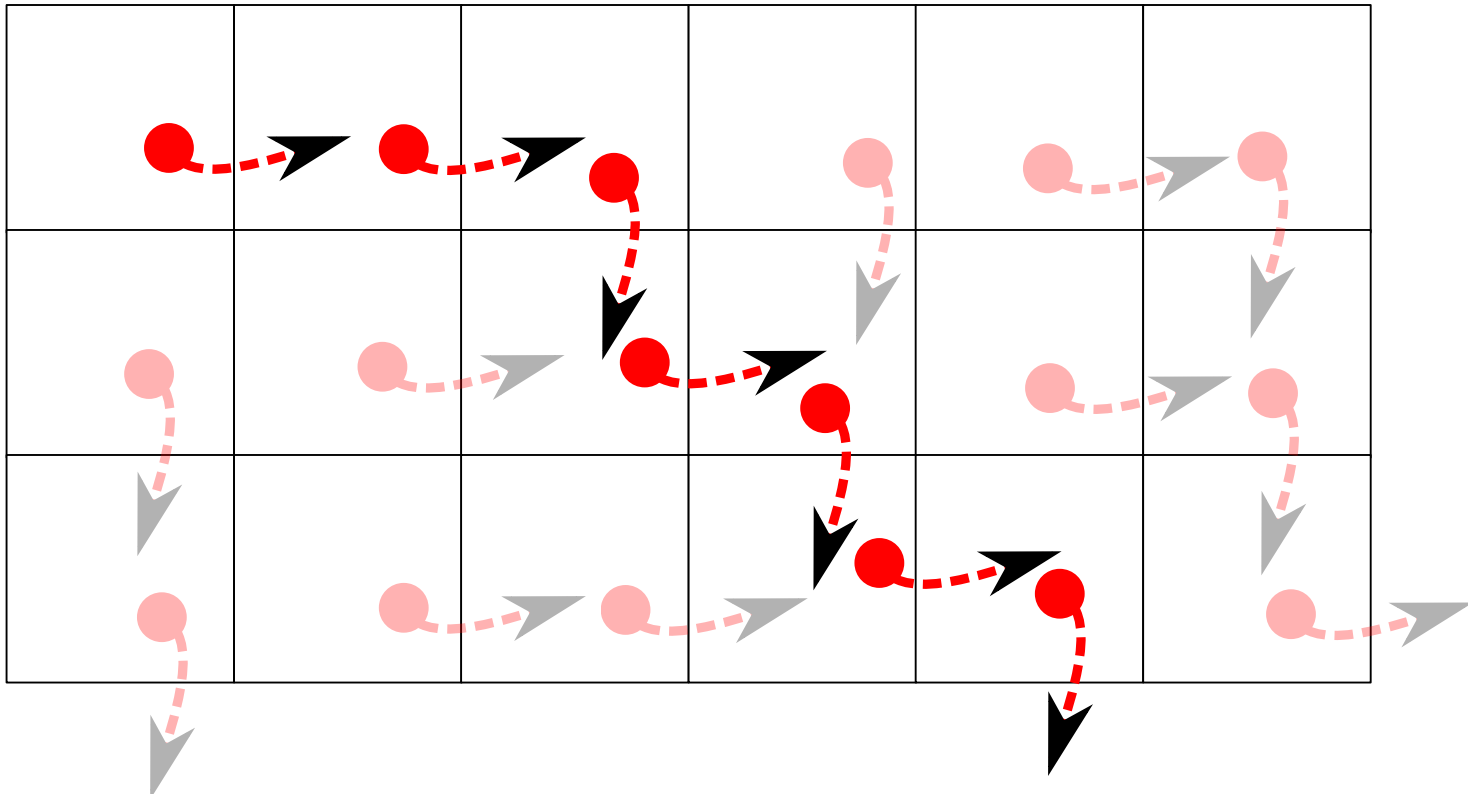
Rascunho de solução

```
  ...
  int opc1 = ...
  int opc2 = ...
  if (opc1 > opc2) {
    maior = opc1;
    escolha[i][j] = 1; // ou sua convensao
  } else {
    maior = opc2;
    escolha[i][j] = 2; // ou sua convensao
  }
  mem[i][j] = maior;
  return maior;
}
```

Na matriz



Caminho que descreve a resposta

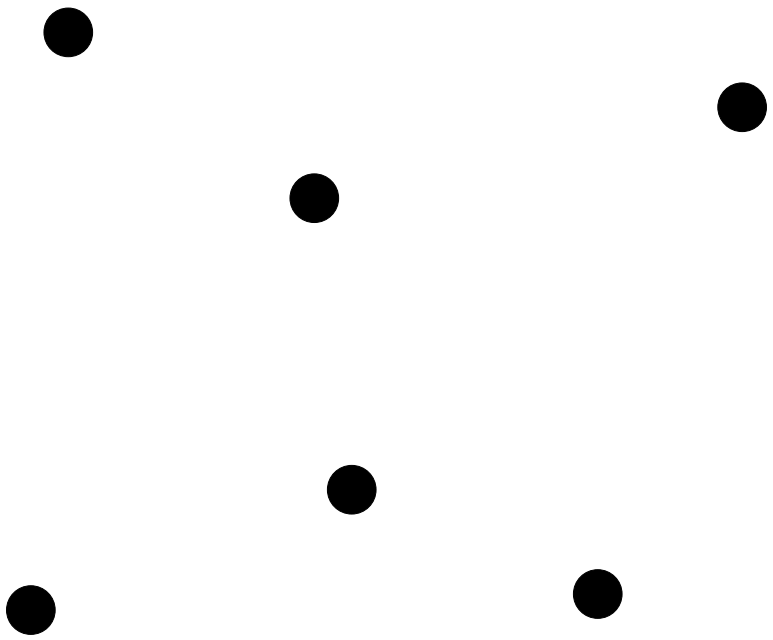


Rascunho de solução

```
i=j=0;
while (!fora_da_matriz(i,j)) {
    if (escolha[i][j]==1) {
        // acao de acordo com a escolha 1
        i = i+1;
    else
        // acao de acordo com a escolha 2
        j = j+1;
    }
}
```

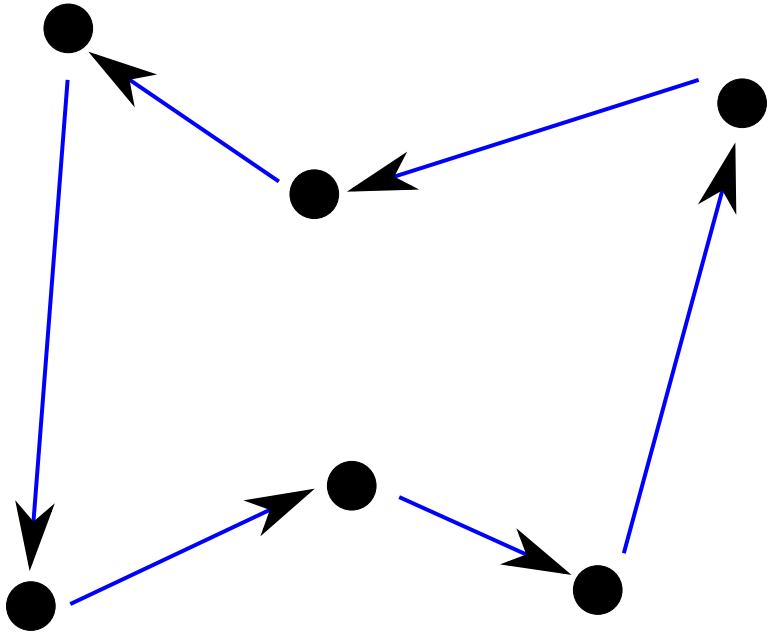
O problema do Caixeiro Viajante

- Euclidiano



O problema do Caixeiro Viajante

- Euclidiano



Recorrência

- v_{ini}: Cidade inicial
- u: Cidade atual
- S: *Conjunto* de cidades visitadas

$$tsp(u, S) = \begin{cases} dist(u, v_{ini}) \text{ se } |S| = N \\ \min\limits_{v \notin S} \left(dist(u, v) + tsp(v, S \cup \{v\})\right) \text{ c.c.} \end{cases}$$

Custo Computacional

N 2^N

$$tsp(u, S) = \begin{cases} dist(u, v_{ini}) \text{ se } |S| = N \\ \max\limits_{v \notin S} \left(dist(u, v) + tsp(v, S \cup \{v\})\right) \text{ c.c.} \end{cases}$$

O(N)

- Espaço
  - O(estados) $= O(2^N \times N)$
- Tempo
  - O(estados) $\times$ O(cada estado)
  - $O(N \times 2^N) \times O(N) = \mathbf{O(2^N \times N^2)}$

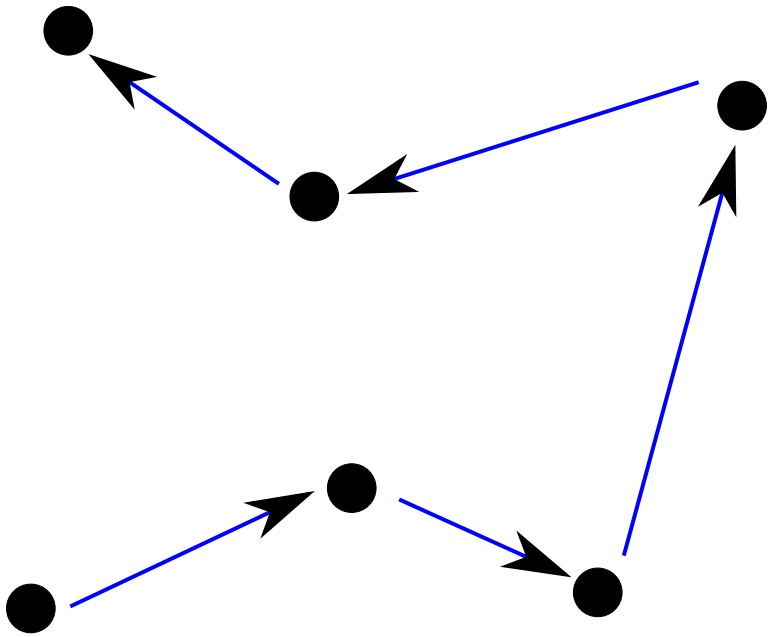
Representação de Conjuntos

- *Bitmask:* `int bm;`
- $v \in S$ sse $v-$ésimo *bit* de `bm` é 1
- $(v \in S)?$: `(bm&(1<<v)) ?`
- $S = \emptyset$: `bm = 0`
- $|S|$: `__builtin_popcount(bm)`
- $|S| = N$: `bm = (1<<N)-1`
- $S = S \cup \{v\}$: `bm = bm | (1<<v)`
- $S = S \setminus \{v\}$: `bm = bm & ~(1<<v)`

Rascunho de solução

```
double tsp(int u, int bm) {
  ...
  double menor = 1e20;
  for (int v = 0;v < n; v++) if (!(bm&(1<<v))) {
    double opc = dist(u, v) + tsp(v, bm | (1<<v));
    menor = min(menor, opc);
  }
  ...
  return menor;
}
```

Caminho Hamiltoniano



Rascunho de solução

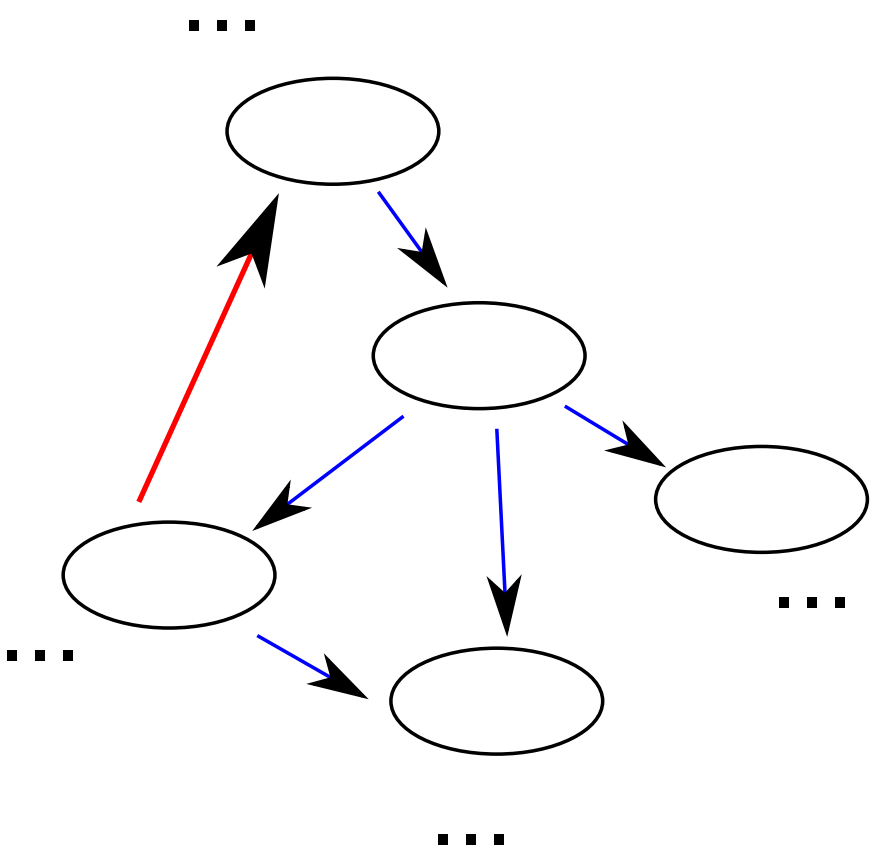
```
int main() {
    ...
    double menor = 1e20;
    for (int prim=0;prim<n;prim++) {
        double opc = tsp(prim, (1<<prim));
        menor = min(menor,opc);
    }
    ...
}
```

Recorrência para Lâmpadas

$$lampada(\{1..N\}) = 0$$
$$lampada(S) = 1 + min($$
$$\{lampada(S \leftrightarrow I) | 0 \leq I < N\} \cup$$
$$\{lampada(S \leftrightarrow I, (I-1)\%N, (I+1)\%N) | 0 \leq I < N\})$$

- onde $S \leftrightarrow I = S \cup \{I\}$ se $I \notin S$, $S \setminus \{I\}$ c.c.

Máquina de estados



Recorrência para Lâmpadas

$$lampada(\{1..N\}) = 0$$
$$lampada(S) = 1 + min($$
$$\{lampada(S \leftrightarrow I) | 0 \leq I < N\} \cup$$
$$\{lampada(S \leftrightarrow I, (I-1)\%N, (I+1)\%N) | 0 \leq I < N\})$$

- #SQN!

Recorrência para Lâmpadas 2

$$
\begin{aligned}
&L2(\{1..N\}) = 0 \\
&L2(S) = \\
&P_0 \times (0.5 \times (1 + L2(S \leftrightarrow 0)) + 0.5 \times (1 + L2(S \leftrightarrow (N-1, 0, 1))))+ \\
&P_1 \times (0.5 \times (1 + L2(S \leftrightarrow 1)) + 0.5 \times (1 + L2(S \leftrightarrow (0, 1, 2))))+ \\
&P_2 \times (0.5 \times (1 + L2(S \leftrightarrow 2)) + 0.5 \times (1 + L2(S \leftrightarrow (1, 2, 3))))+ \\
&...+ \\
&P_{N-1} \times (0.5 \times (1 + L2(S \leftrightarrow N-1)) + 0.5 \times (1 + L2(S \leftrightarrow \\
&N-2, N-1, 0)))
\end{aligned}
$$

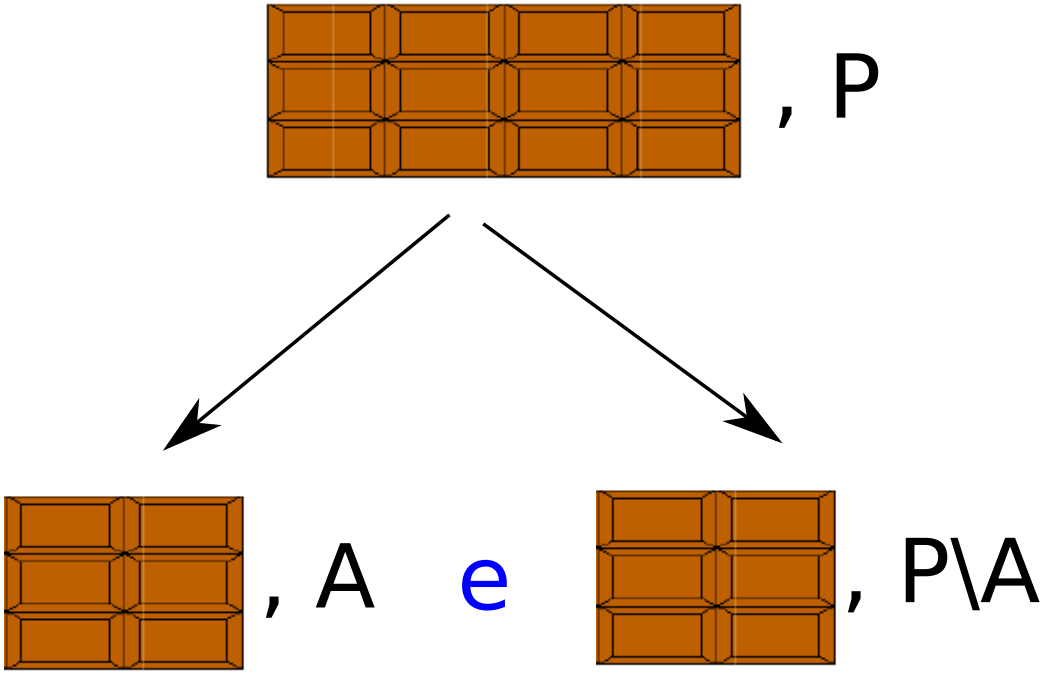
Xunxo: MAXDEPTH

- Ponto-chave: Observar se o fator multiplicando a recorrência eventualmente converge para zero!
- Força um grafo acíclico neste caso.
- **Arriscado!** Use por sua conta e risco!

Recorrência **Xunxada** para Lâmpadas 2

$L2(\{1..N\},\mathrm{d}) = 0$
$L2(S, MAXDEPTH) = 0$
$L2(S,\mathrm{d}) =$
$P_0 \times (0.5 \times (1 + L2(S \leftrightarrow 0,\mathrm{d}+1) + 0.5 \times (1 + L2(S \leftrightarrow$
$(N-1,0,1),\mathrm{d}+1)))+$
$P_1 \times (0.5 \times (1 + L2(S \leftrightarrow 1,\mathrm{d}+1) + 0.5 \times (1 + L2(S \leftrightarrow (0,1,2),\mathrm{d}+1)))+$
$P_2 \times (0.5 \times (1 + L2(S \leftrightarrow 2,\mathrm{d}+1) + 0.5 \times (1 + L2(S \leftrightarrow (1,2,3),\mathrm{d}+1)))+$
$...+$
$P_{N-1} \times (0.5 \times (1 + L2(S \leftrightarrow N-1,\mathrm{d}+1)) + 0.5 \times (1 + L2(S \leftrightarrow$
$(N-2,N-1,0),\mathrm{d}+1)))$

Força Bruta para Chocolate



Rascunho de solução

```
bool chocolate(int L, int C, int P_bm) {
    ...
    // corte horizontal
    for (int L1=1;L1<L;L1++)
        // para cada subconjunto A de P_bm
        for(...)
            if (chocolate(L1,C,A) and
                chocolate(L-L1,C,P_bm-A))
                  return true;

    // corte vertical
    ...
    return false;
}
```

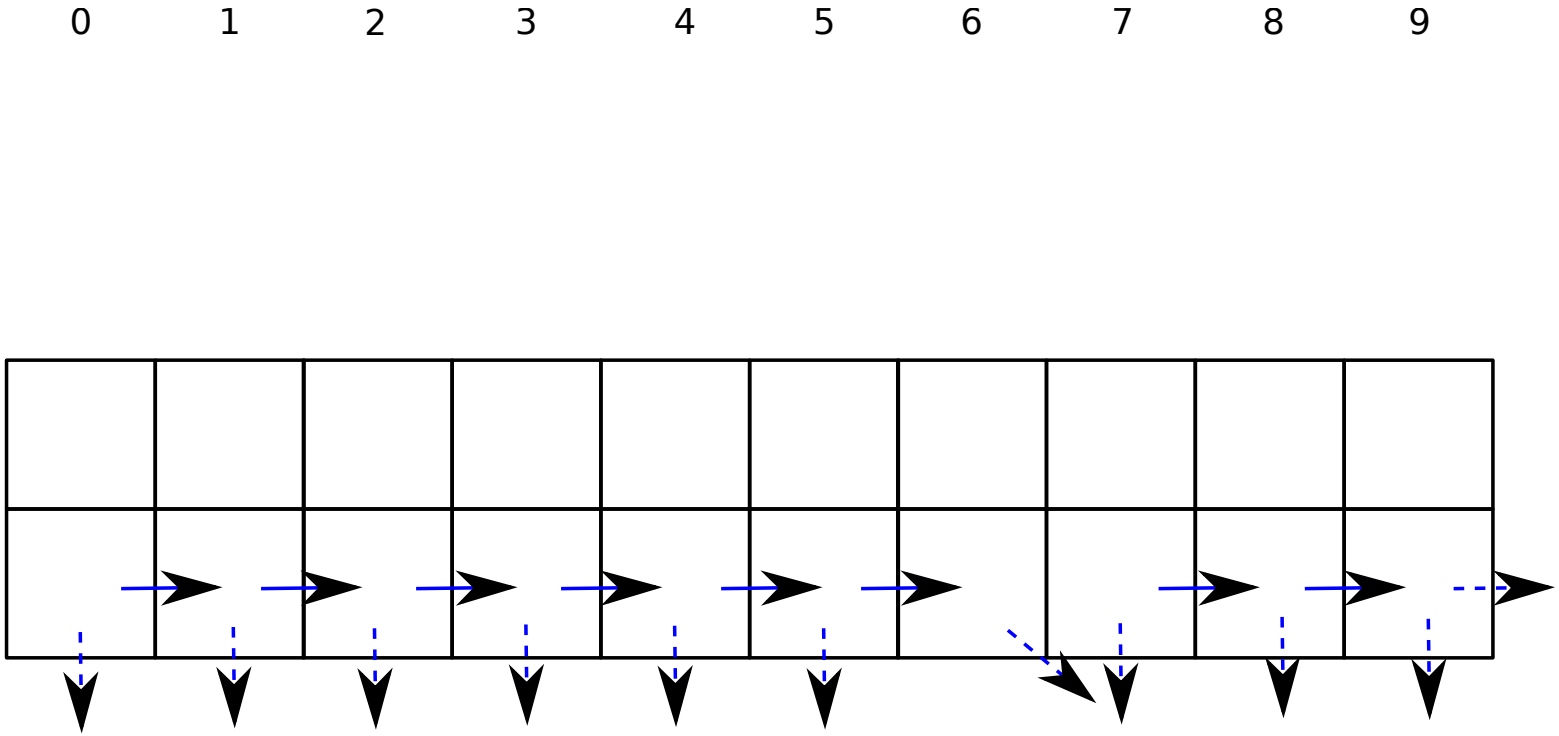
Redução de estado

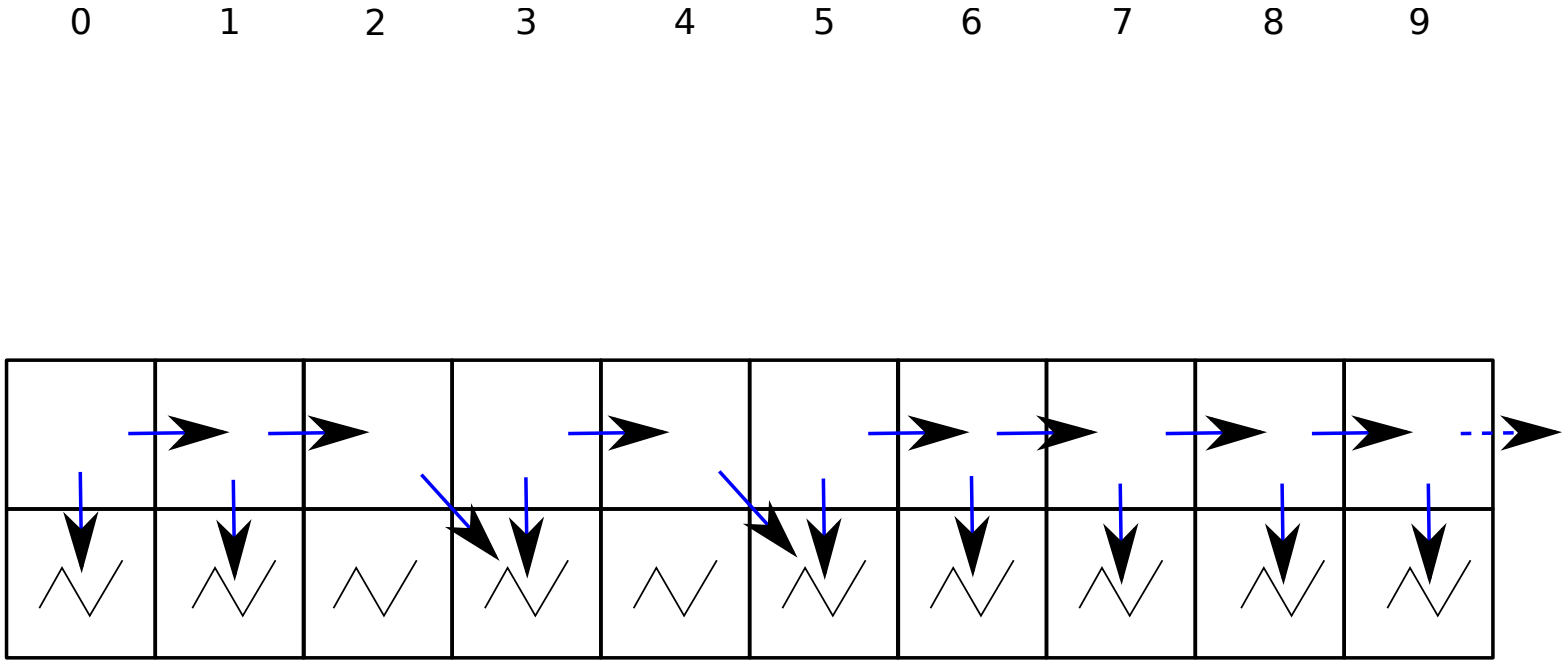
- Quando um parâmetro do estado puder ser obtido em função de outros, este pode ser removido do estado!

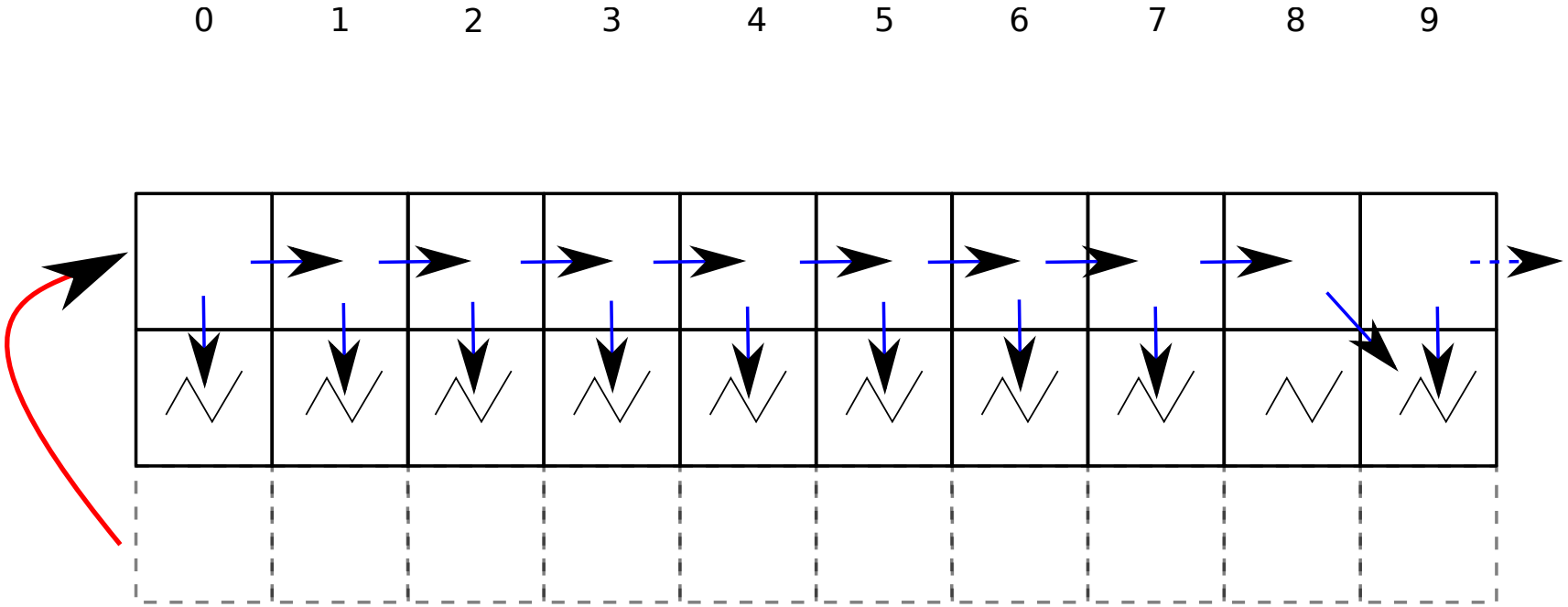
```
bool chocolate(int L, int P_bm) {
    int C = L/soma[P_bm];

    ...
```

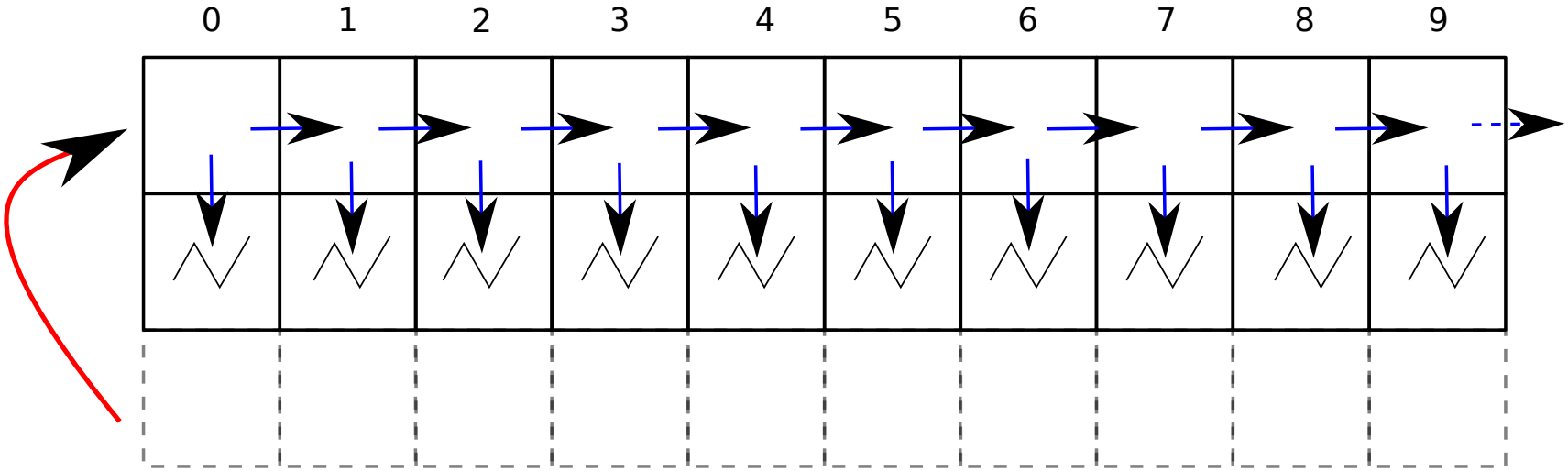
Otimização de espaço

- *Ás vezes*, o espaço pode ser reduzido
- Verificar matriz de dependência
- Mochila
  - de $O(N \times C)$ para $O(C)$
- LCS
  - de $O(N \times M)$ para $O(M)$


0
1
2
3
4
5
6
7
8
9


0
1
2
3
4
5
6
7
8
9


0
1
2
3
4
5
6
7
8
9


0
1
2
3
4
5
6
7
8
9

Rascunho de solução

```
int P[2][1024];

...
int atu=0;

for (int i=...) {
  for (int j=...) {
    ...
    PD[atu][j] = max(PD[atu][j+1], PD[atu^1][j]);
    ...
  }
  atu = atu^1; // troca o papel das linhas
}
```

Zerar/Inicializar uma matriz em $O(1)$

- Múltiplas execuções de PDs em um único caso de teste
- Entrada com múltiplos casos de teste
- Percorre a matriz uma única vez
- Demais inicializações em $O(1)$

Timestamps

```
int timestamp[1024][1024], TS;
int calc(int i, int j) {
    if (timestamp[i][j]==TS)
        return mem[i][j];
    ...
    timestamp[i][j]=TS;
    mem[i][j] = maior;
}
int main() {
    memset(timestamp,0,sizeof(timestamp));
    TS=0;
    ...
    TS++;   // Zera matriz
    int resp = calc(0,0);
}
```

Compactação de Coordenadas simplificada

```
typedef pair<int,int> ii;
map<ii, int> PD;
map<ii, int>::iterator PDi;
...
int funcao(int a, int b) {
    ...
    PDi = PD.find(ii(a,b));
    if (PDi != PD.end())
        return PDi->second;
    ...
    return PD[ii(a,b)] = resp;
}
```

Pratique!

- *http://br.spoj.pl/problems/DESCULPA/*
- *http://br.spoj.com/problems/DOCE/*
- *http://br.spoj.com/problems/HEARTPMG/*
- *http://br.spoj.com/problems/HYPER10/*
- *http://br.spoj.com/problems/NEFER/*
- *https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=44&page=show_problem&problem=2795*
- *https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=3069*
- **Todos** em *http://ahmed-aly.com/Category.jsp?ID=33*
- **Maioria** de *https://www.urionlinejudge.com.br/judge/pt/problems/index/6*