ACH2002

Aula 12

Técnicas de Desenvolvimento de Algoritmos - **Algoritmos Gulosos (Greedy algorithms)**

Profa. Ariane Machado Lima

(adaptado dos slides de aula da Profa. Fátima L. S. Nunes)

Aulas passadas

- Técnicas de programação:
  - **Divisão e conquista**:
  - **Programação dinâmica**:

Aulas passadas

- Técnicas de programação:
  - **Divisão e conquista**: solução do problema original pode ser obtida combinando soluções de subproblemas (indução fraca) – vai bem com subproblemas disjuntos
  - **Programação dinâmica**: solução do problema original pode ser obtida combinando soluções de subproblemas que se sobrepõem – para evitar reexecuções precisa armazenar soluções em uma estrutura de dados (normalmente uma matriz) (indução forte)

EACH USP

Aula de hoje

- Há muitos casos, em problemas de otimização, que usar técnicas sofisticadas como programação dinâmica é um tiro que canhão para matar um rato
- Que estratégias mais simples poderiam funcionar?
- Algoritmos **gulosos** (dependendo do problema)

Algoritmos gulosos

- O que é guloso?

Algoritmos gulosos

- O que é guloso?

adj. e s.m. Que ou quem come muito; comilão; glutão

Quer SEMPRE o maior pedaço

No caso de um algoritmo, toma sempre (em cada passo) a melhor decisão para aquele passo, sem olhar para os demais passos.

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo: o problema do GPS
  - queremos encontrar o melhor caminho entre dois locais

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo: o problema do GPS
  - queremos encontrar o melhor caminho entre dois locais:
    - mais curto;
    - mais barato (menos pedágio);
    - mais rápido;
    - mais bonito.

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Existem soluções **viáveis** (uma possível) e soluções ótimas (as melhores de todas segundo algum critério)
- Exemplo: o problema do GPS
  - Queremos encontrar um caminho entre dois locais (soluções viáveis)
  - queremos encontrar o melhor caminho entre dois locais (soluções ótimas). Possíveis critérios de otimalidade:
    - mais curto;
    - mais barato (menos pedágio);
    - mais rápido;
    - mais bonito.

EACH USP

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo: o problema do GPS
  - queremos encontrar o melhor caminho entre dois locais:
    - mais curto;

Como podemos representar as características dos caminhos entre cidades vizinhas com o que sabemos até agora?

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo: o problema do GPS
  - queremos encontrar o melhor caminho entre dois locais:
    - mais curto;

Como podemos representar as características dos caminhos entre cidades vizinhas com o que sabemos até agora?

Matrizes!

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo: considerando que cada célula da matriz é a distância entre duas cidades vizinhas, qual é o caminho mais curto entre as cidades A e B?

A	4	11	1	2	9	8
9	3	20	14	5	7	7
10	2	5	6	6	6	6
15	10	12	3	8	5	3
2	6	22	2	9	5	B
5	2	11	9	3	7	8

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo: considerando que cada célula da matriz é a distância entre duas cidades vizinhas, qual é o caminho mais curto entre as cidades A e B?

A	4	11	1	2	9	8
9	3	20	14	5	7	7
10	2	5	6	6	6	6
15	10	12	3	8	5	3
2	6	22	2	9	5	B
5	2	11	9	3	7	8

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo: considerando que cada célula da matriz é a distância entre duas cidades vizinhas, qual é o caminho mais curto entre as cidades A e B?

A	4	11	1	2	9	8
9	3	20	14	5	7	7
10	2	5	6	6	6	6
15	10	12	3	8	5	3
2	6	22	2	9	5	B
5	2	11	9	3	7	8

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo: considerando que cada célula da matriz é a distância entre duas cidades vizinhas, qual é o caminho mais curto entre as cidades A e B?

A	4	11	1	2	9	8
9	3	20	14	5	7	7
10	2	5	6	6	6	6
15	10	12	3	8	5	3
2	6	22	2	9	5	B
5	2	11	9	3	7	8

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo: considerando que cada célula da matriz é a distância entre duas cidades vizinhas, qual é o caminho mais curto entre as cidades A e B?

A	4	11	1	2	9	8
9	3	20	14	5	7	7
10	2	5	6	6	6	6
15	10	12	3	8	5	3
2	6	22	2	9	5	B
5	2	11	9	3	7	8

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo: considerando que cada célula da matriz é a distância entre duas cidades vizinhas, qual é o caminho mais curto entre as cidades A e B?

A	4	11	1	2	9	8
9	3	20	14	5	7	7
10	2	5	6	6	6	6
15	10	12	3	8	5	3
2	6	22	2	9	5	B
5	2	11	9	3	7	8

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo: considerando que cada célula da matriz é a distância entre duas cidades vizinhas, qual é o caminho mais curto entre as cidades A e B?

A	4	11	1	2	9	8
9	3	20	14	5	7	7
10	2	5	6	6	6	6
15	10	12	3	8	5	3
2	6	22	2	9	5	B
5	2	11	9	3	7	8

Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo: considerando que cada célula da matriz é a distância entre duas cidades vizinhas, qual é o caminho mais curto entre as cidades A e B?

A	4	11	1	2	9	8
9	3	20	14	5	7	7
10	2	5	6	6	6	6
15	10	12	3	8	5	3
2	6	22	2	9	5	B
5	2	11	9	3	7	8

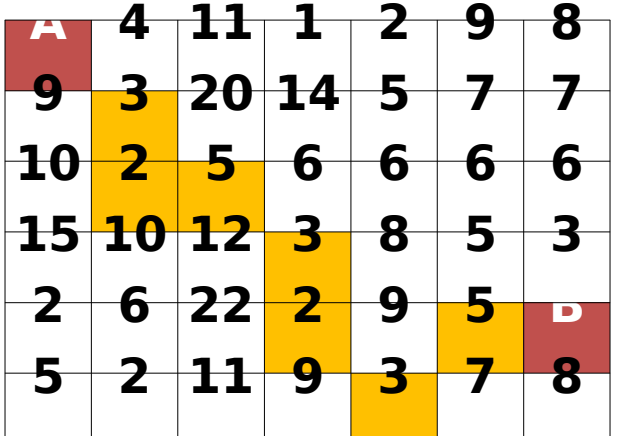
Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Exemplo:
  - encontrar o menor caminho entre duas cidades A e B;
  - digamos que pode-se escolher qualquer direção;
  - cada célula contém a distância da cidade atual até a próxima cidade.

A	4	11	1	2	9	8
9	3	20	14	5	7	7
10	2	5	6	6	6	6
15	10	12	3	8	5	3
2	6	22	2	9	5	B
5	2	11	9	3	7	8

Algoritmos gulosos

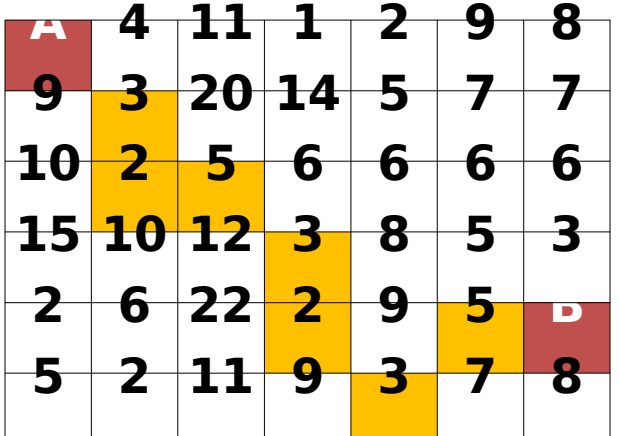
- Tipicamente usados para resolver problemas de otimização.
- Em cada passo:
  - escolhe a decisão ótima em cada passo, na esperança de obter solução ótima global *(mas nem sempre consegue!)*;
  - nunca reconsidera a decisão tomada em um momento anterior;



Algoritmos gulosos

- Tipicamente usados para resolver problemas de otimização.
- Em cada passo:
  - escolhe a decisão ótima em cada passo, na esperança de obter solução ótima global *(mas nem sempre consegue!)*;
  - nunca reconsidera a decisão tomada em um momento anterior;

- uma vez que o candidato é adicionado à solução, permanecerá na solução para sempre;
- uma vez que o candidato é rejeitado, nunca mais será considerado.



Algoritmos gulosos

- Algoritmo guloso é eficiente para uma grande variedade de problemas.
- Vamos analisar dois problemas:
  - escolher produtos que caibam em uma mochila (famoso ‘problema da mochila’).
  - locação de atividades em uma sala;

Algoritmos gulosos

- Exemplo 1: problema da mochila (valor)

- há uma mochila que admite um peso máximo
- há um conjunto de objetos, cada um com um valor e um peso;
- devemos selecionar o conjunto de objetos que caibam dentro da mochila de forma a maximizar o valor total dentro dela.

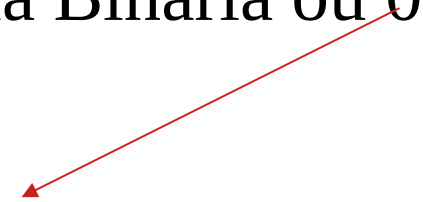
Notem uma pequena variação do problema da mochila visto na aula passada:

- lá queria-se maximizar o peso (e tinha que ser exatamente o tamanho da mochila)
- aqui se quer maximizar o valor, e não precisa encher a mochila até a sua capacidade total

Algoritmos gulosos

- Exemplo 1: problema da mochila

- Dois subproblemas distintos:
  - Objetos podem ser particionados (e o valor será proporcional à fração do objeto), ou seja, você pode colocar um pedaço do objeto dentro da mochila;
    - Ex: ouro em pó
  - Os objetos não podem ser particionados (ou estarão dentro da mochila ou fora). Problema conhecido como "Problema da Mochila Binária ou 0-1"
    - Ex: ouro em barras



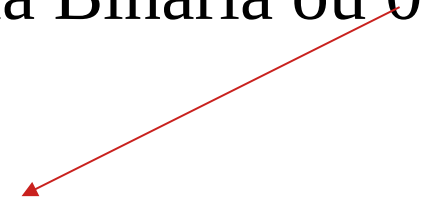
Resolvemos a aula passada usando programação dinâmica
Que característica era complicada neste problema?

EACH USP

Algoritmos gulosos

- Exemplo 1: problema da mochila

- Dois subproblemas distintos:
  - Objetos podem ser particionados (e o valor será proporcional à fração do objeto), ou seja, você pode colocar um pedaço do objeto dentro da mochila;
    - Ex: ouro em pó
  - Os objetos não podem ser particionados (ou estarão dentro da mochila ou fora). Problema conhecido como "Problema da Mochila Binária ou 0-1"
    - Ex: ouro em barras



Resolvemos a aula passada usando programação dinâmica
Que característica era complicada neste problema?
Várias combinações tinham que ser testadas!

Algoritmos gulosos

- Exemplo 1: problema da mochila

- Dois subproblemas distintos:
  - Objetos podem ser particionados (e o valor será proporcional à fração do objeto), ou seja, você pode colocar um pedaço do objeto dentro da mochila;
    - Ex: ouro em pó

Será que esse problema é assim tão complicado?

  - Os objetos não podem ser particionados (ou estarão dentro da mochila ou fora). Problema conhecido como "Problema da Mochila Binária ou 0-1"
    - Ex: ouro em barras

Resolvemos a aula passada usando programação dinâmica
Que característica era complicada neste problema?
Várias combinações tinham que ser testadas!

Algoritmos gulosos

- Exemplo 1: problema da mochila

- Problema da mochila fracionada:
  - Qual seria a solução?

Algoritmos gulosos

- Exemplo 1: problema da mochila

- Problema da mochila fracionada:
  - Qual seria a melhor ordenação da entrada?

Algoritmos gulosos

- Exemplo 1: problema da mochila

- Problema da mochila fracionada:
  - Qual seria a melhor ordenação da entrada?
    - ordenar pelo valor/peso
  - Algoritmo:

Algoritmos gulosos

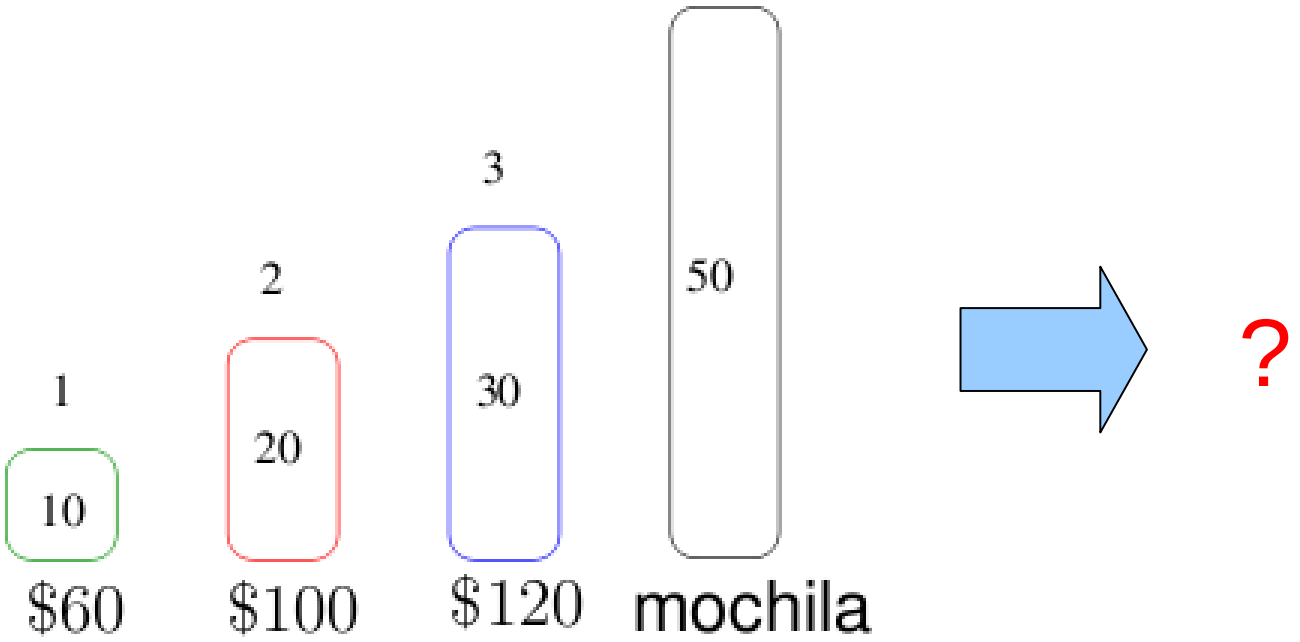
- Exemplo 1: problema da mochila

- Problema da mochila fracionada:
  - Qual seria a melhor ordenação da entrada?
    - ordenar pelo valor/peso
  - A solução gulosa será ótima?
    - Sim (é demonstrável)
  - Algoritmo:
    - ordenar os itens por valor/peso decrescentemente;
    - colocar na mochila o máximo do item *i* que estiver disponível e for possível;
    - passar para o próximo item.

Algoritmos gulosos

- Exemplo 1: problema da mochila

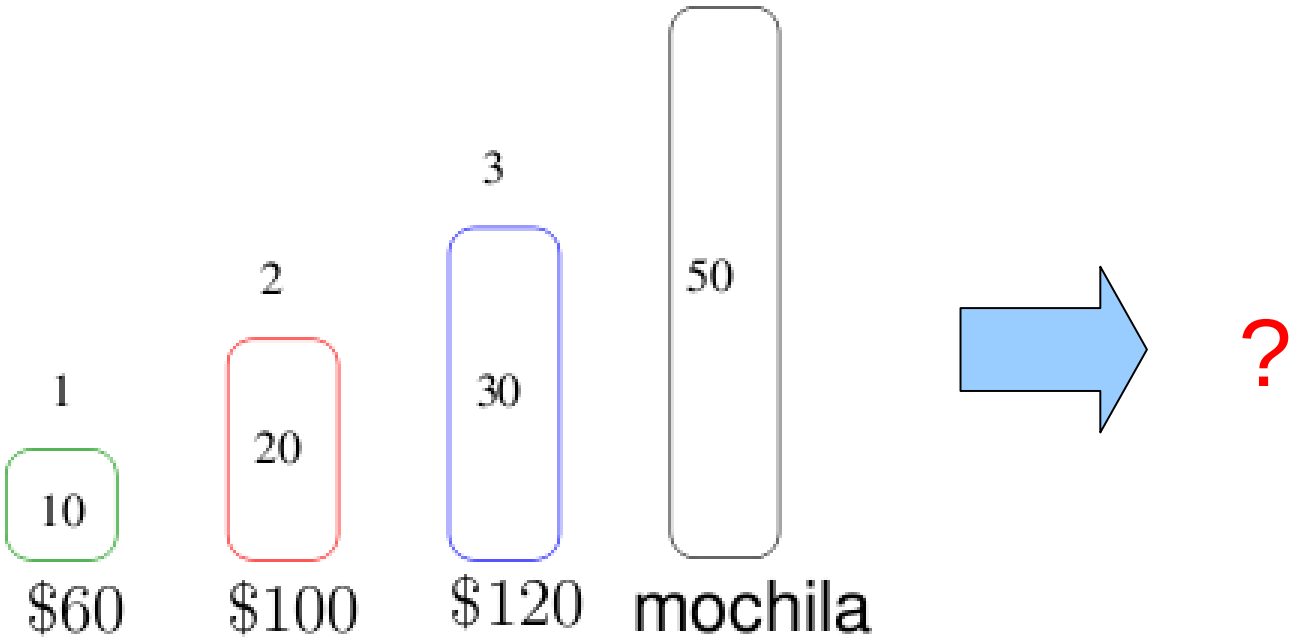
- Problema da mochila fracionada:



Algoritmos gulosos

- Exemplo 1: problema da mochila

- Problema da mochila fracionada:

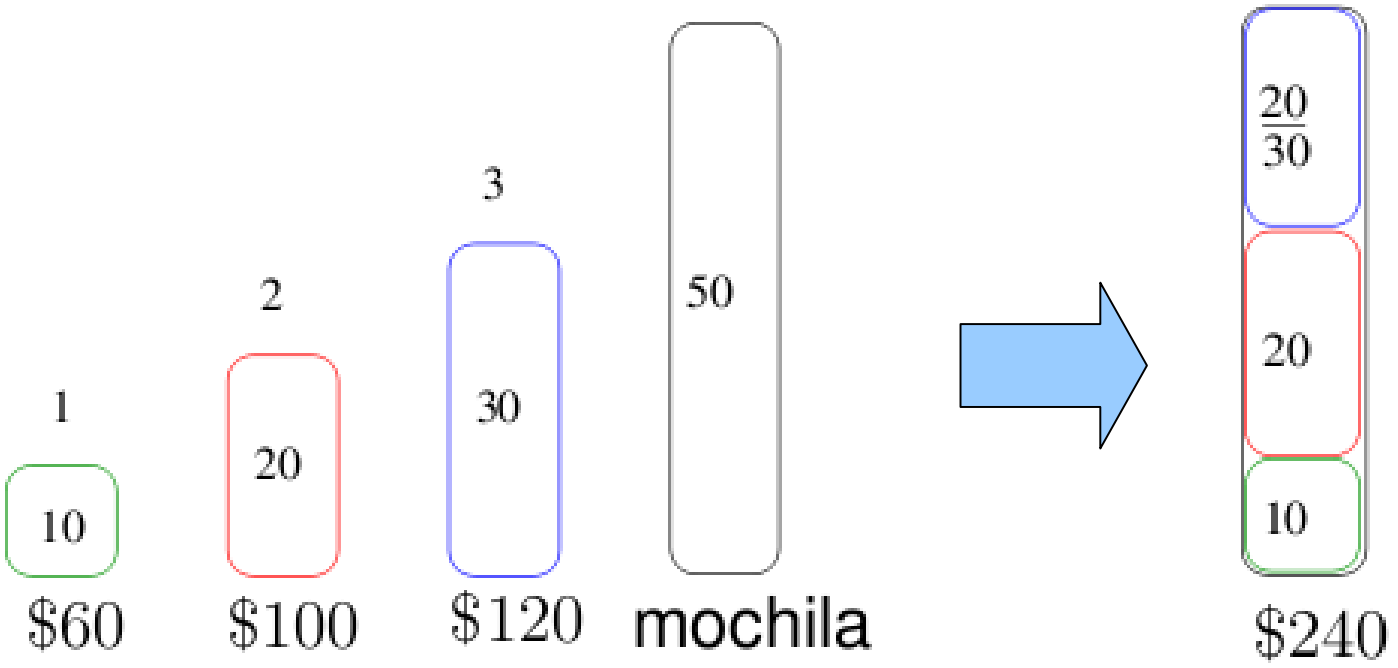


valor/peso = 6 5 4

Algoritmos gulosos

- Exemplo 1: problema da mochila

- Problema da mochila fracionada:



valor/peso = 6 5 4

Algoritmos gulosos

- Exemplo 1: problema da mochila

```
/* Entrada: K = capacidade máxima da mochila
            n = nr de itens
            v = vetor de valores dos n itens
            p = vetor de pesos dos n itens
   Saída: vetor f em que f[i] contém a fração do item i na mochila */
mochilaFracionada(K, n, v, p)
Crie um um vetor I dos n itens ordenados descrescentemente pelo valor/peso;
capacidade ← K // capacidade atual da mochila
i ← 1
enquanto i <= n  e  capacidade >= p[i]    faça
   f[i] ← 1 // Pegue todo o item i
   capacidade ← capacidade – p[i]
   i ← i + 1
se i <= n //provavelmente ainda tem espaço, embora não caiba i-ésimo item todo
   f[i] ← capacidade/p[i] // Pegue só o que der do item i
para j ← i+1 até n
   f[j] ← 0    //dos outros itens não dá para pegar nada
retorna f
```

Algoritmos gulosos

- Exemplo 1: problema da mochila

```
/* Entrada: K = capacidade máxima da mochila
            n = nr de itens
            v = vetor de valores dos n itens
            p = vetor de pesos dos n itens
   Saída: vetor f em que f[i] contém a fração do item i na mochila */
mochilaFracionada(K, n, v, p)
Crie um um vetor I dos n itens ordenados descrescentemente pelo valor/peso;
capacidade ← K // capacidade atual da mochila
i ← 1
enquanto i <= n  e  capacidade >= p[i]    faça
   f[i] ← 1 // Pegue todo o item i
   capacidade ← capacidade – p[i]
   i ← i + 1
se i <= n //provavelmente ainda tem espaço, embora não caiba i-ésimo item todo
   f[i] ← capacidade/p[i] // Pegue só o que der do item i
para j ← i+1 até n
   f[j] ← 0    //dos outros itens não dá para pegar nada
retorna f
```

Complexidade:

Algoritmos gulosos

- Exemplo 1: problema da mochila

```
/* Entrada: K = capacidade máxima da mochila
            n = nr de itens
            v = vetor de valores dos n itens
            p = vetor de pesos dos n itens
   Saída: vetor f em que f[i] contém a fração do item i na mochila */
mochilaFracionada(K, n, v, p)
Crie um um vetor I dos n itens ordenados descrescentemente pelo valor/peso;
capacidade ← K // capacidade atual da mochila
i ← 1
enquanto i <= n  e  capacidade >= p[i]    faça
   f[i] ← 1 // Pegue todo o item i
   capacidade ← capacidade – p[i]
   i ← i + 1
se i <= n //provavelmente ainda tem espaço, embora não caiba i-ésimo item todo
   f[i] ← capacidade/p[i] // Pegue só o que der do item i
para j ← i+1 até n
   f[j] ← 0    //dos outros itens não dá para pegar nada
retorna f
```

Complexidade: O(n) + O(?) + O(n) = O(?) do tempo do alg de ordenação usado

Algoritmos gulosos

- Exemplo 1: problema da mochila

  - Problema da mochila fracionada: prova de corretude

https://www.youtube.com/watch?v=CRYjHV_29gU

Ou

https://www.youtube.com/watch?v=bmGG88LVoY4

Algoritmos gulosos

- Exemplo 1: problema da mochila

- Problema da mochila **binária**:
  - Qual seria a melhor ordenação da entrada?

Algoritmos gulosos

- Exemplo 1: problema da mochila

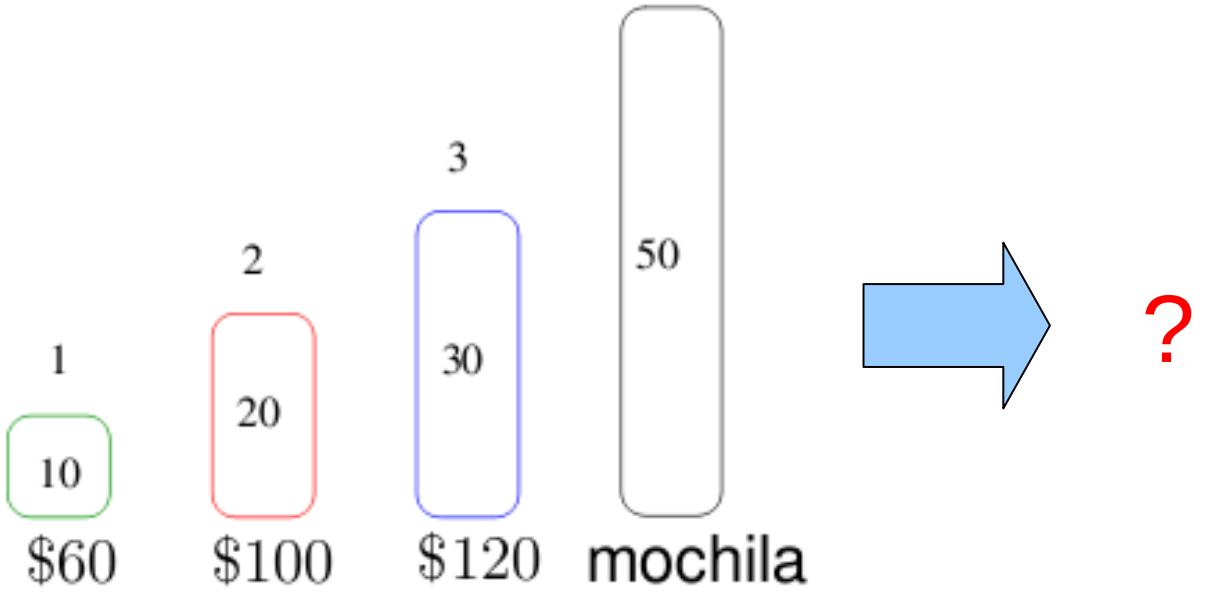
Problema da mochila **binária**:

- Qual seria a melhor ordenação da entrada?
  - ordenar pelo valor/peso
- Algoritmo:
  - ordenar os itens por valor/peso decrescentemente;
  - colocar na mochila o item *i* se for possível;
  - passar para o próximo item.
- A solução gulosa será ótima?

Algoritmos gulosos

- Exemplo 1: problema da mochila

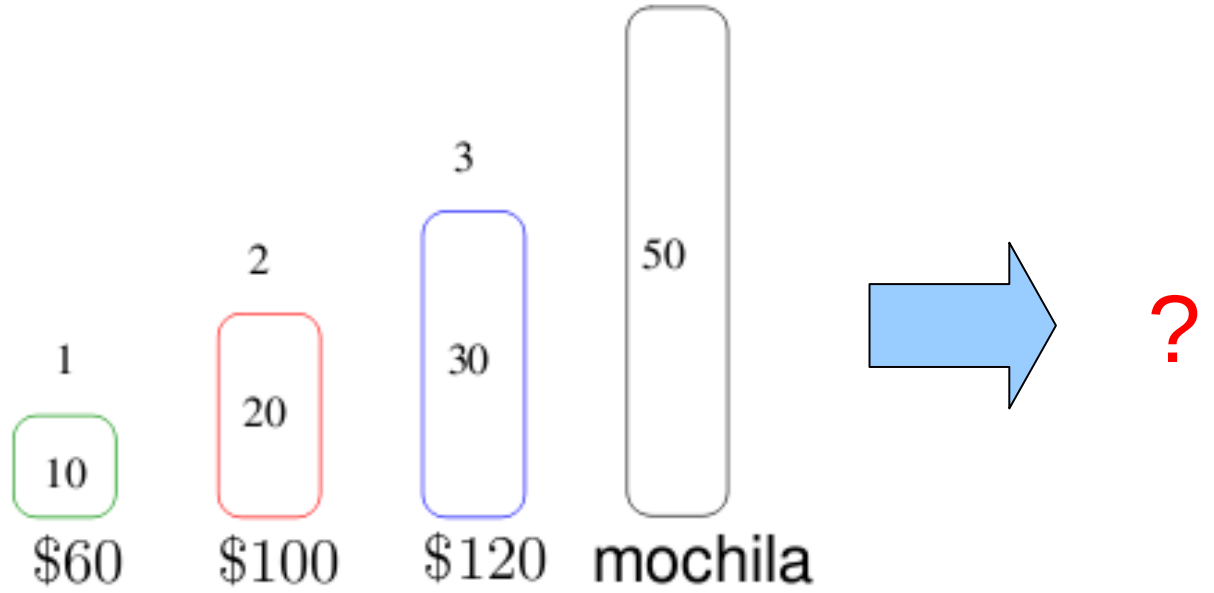
- Problema da mochila **binária**:



Algoritmos gulosos

- Exemplo 1: problema da mochila

- Problema da mochila **binária**:



valor/peso = 6 5 4

Algoritmos gulosos

- Exemplo 1: problema da mochila

- Problema da mochila **binária**:

Algoritmos gulosos

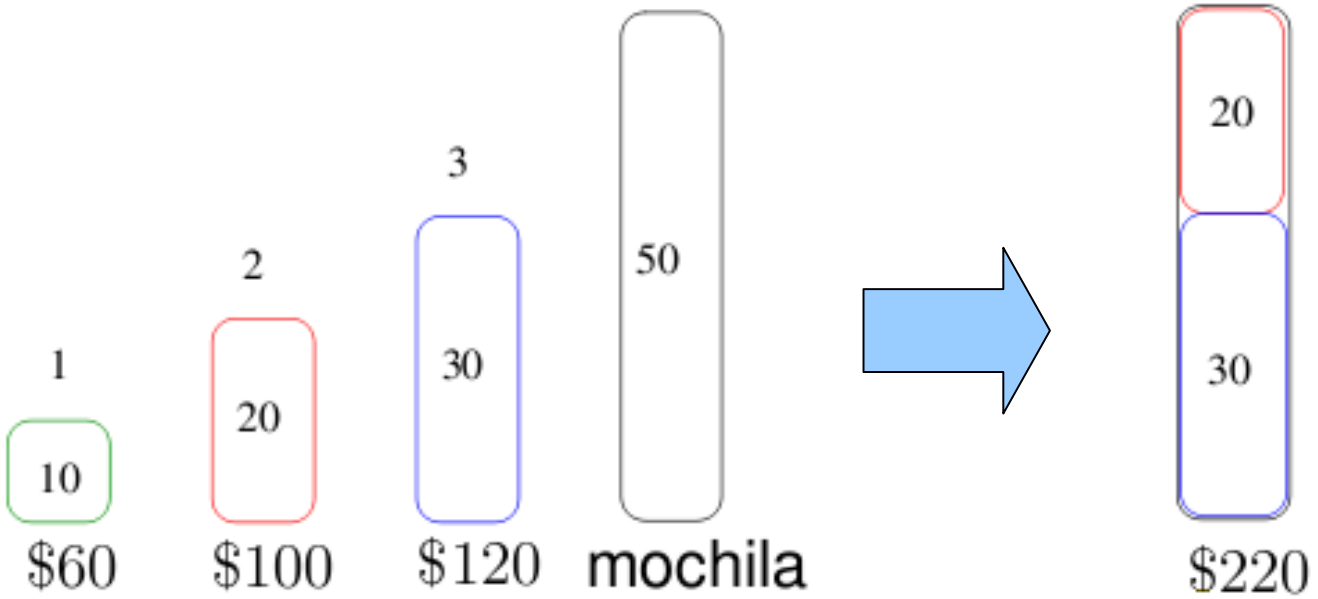
- Exemplo 1: problema da mochila

- Problema da mochila **binária**:
  - A solução gulosa foi ótima?

Algoritmos gulosos

- Exemplo 1: problema da mochila

- Problema da mochila **binária**:
  - A solução gulosa foi ótima?
    - Obviamente não. A ótima seria:



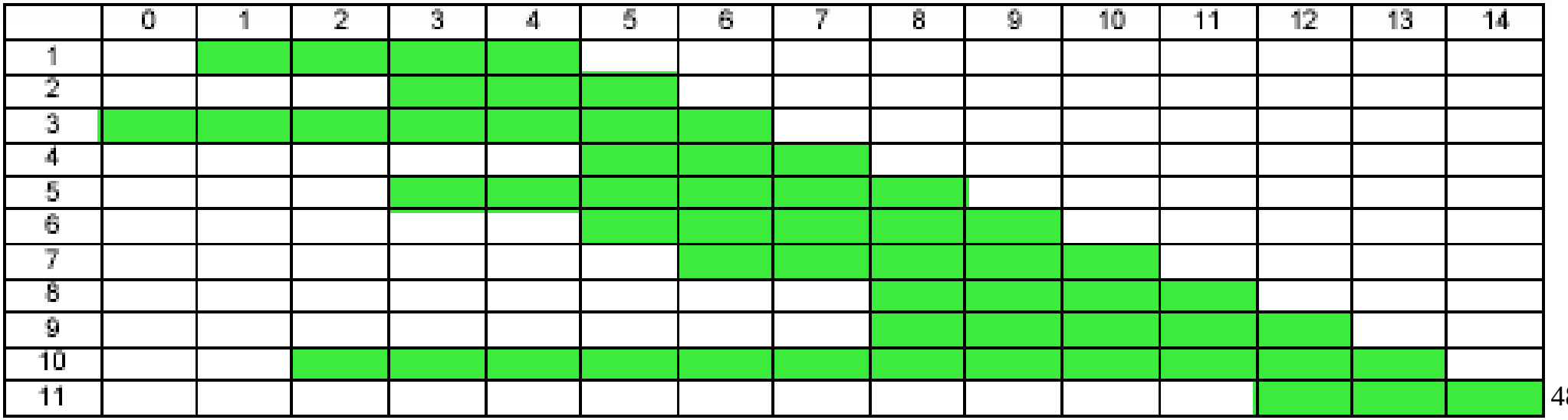
valor/peso = 6 5 4

Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala
  - existem diversas atividades (por exemplo aulas) que querem usar uma mesma sala;
  - cada atividade tem um horário de início e um horário de fim;
  - só existe uma sala disponível;
  - duas aulas não podem ser ministradas na mesma sala ao mesmo tempo.

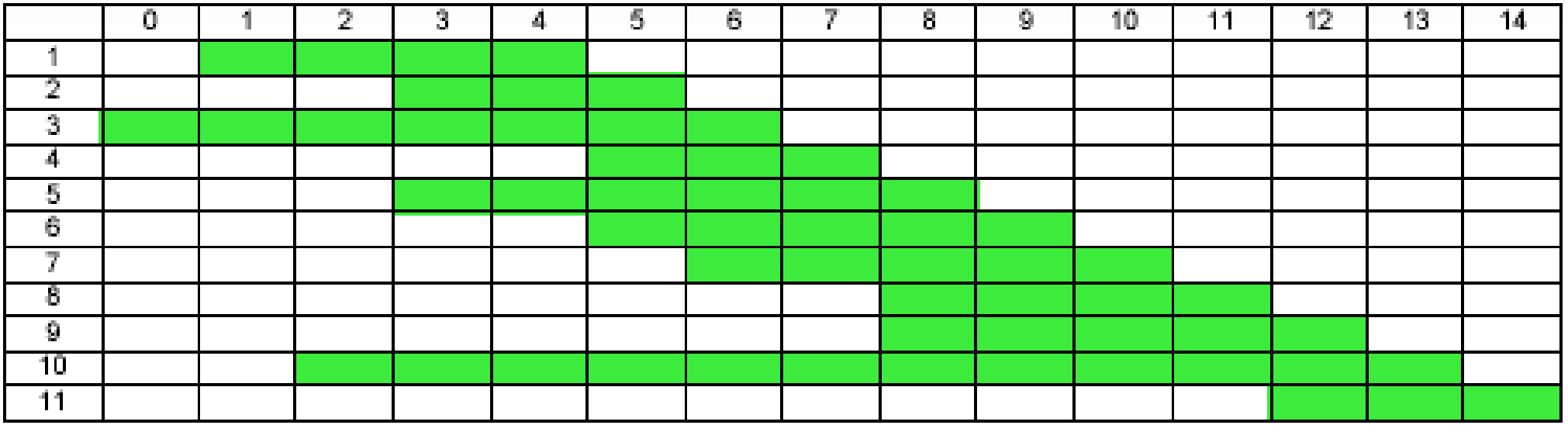
Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala
  - 11 atividades a serem distribuídas em 15 unidades de tempo
  - queremos **selecionar** um conjunto máximo de atividades que não têm sobreposição de tempo
  - Notem que a solução é criada em passos (cada passo é uma seleção de uma atividade
  - O que são soluções viáveis?
  - O que são soluções ótimas?
  - Como resolver?



Algoritmos gulosos

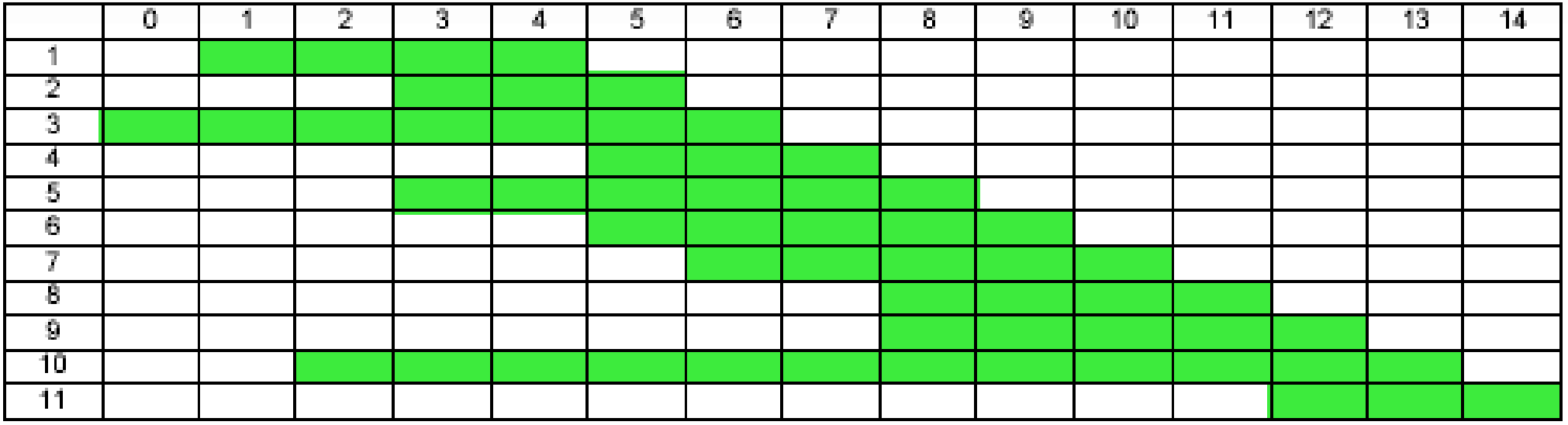
- Exemplo 2: locação de atividades em uma sala
  - 11 atividades a serem distribuídas em 15 unidades de tempo
  - queremos **selecionar** um conjunto máximo de atividades que não têm sobreposição de tempo
  - Notem que a solução é criada em passos (cada passo é uma seleção de uma atividade
  - O que são soluções viáveis?
    - Qualquer conjunto de atividades
  - O que são soluções ótimas?
    - Conjunto máximo de atividades que não se sobrepõem no tempo
  - Como resolver?



Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

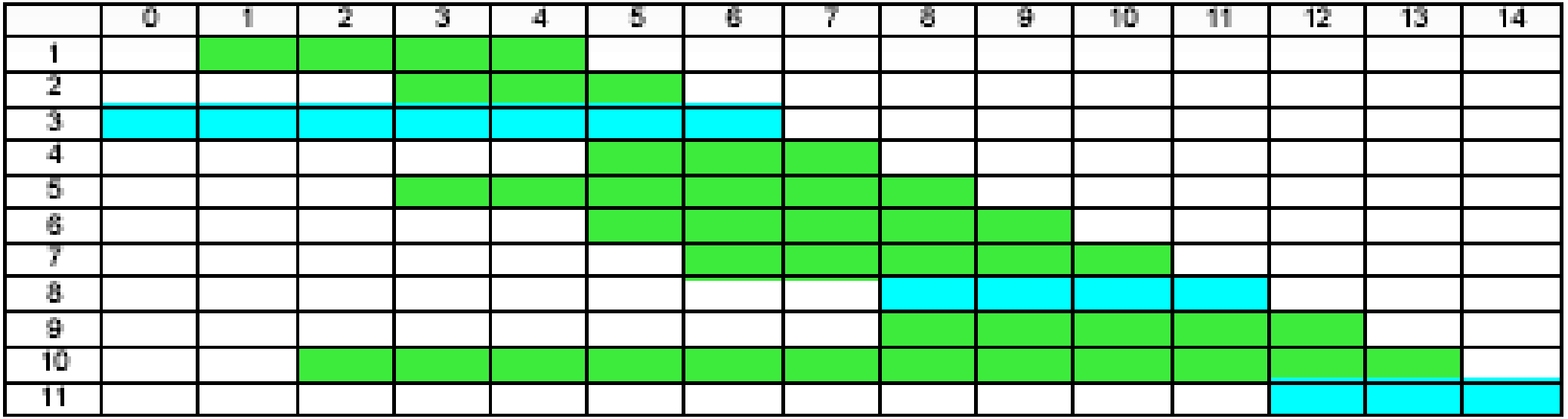
  - Fundamental decidir bem como selecionar:
    - começar pelas atividades que começam primeiro
    - começar pelas atividades que terminam primeiro
    - começar pelas atividades mais longas
    - começar pelas atividades mais curtas



Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

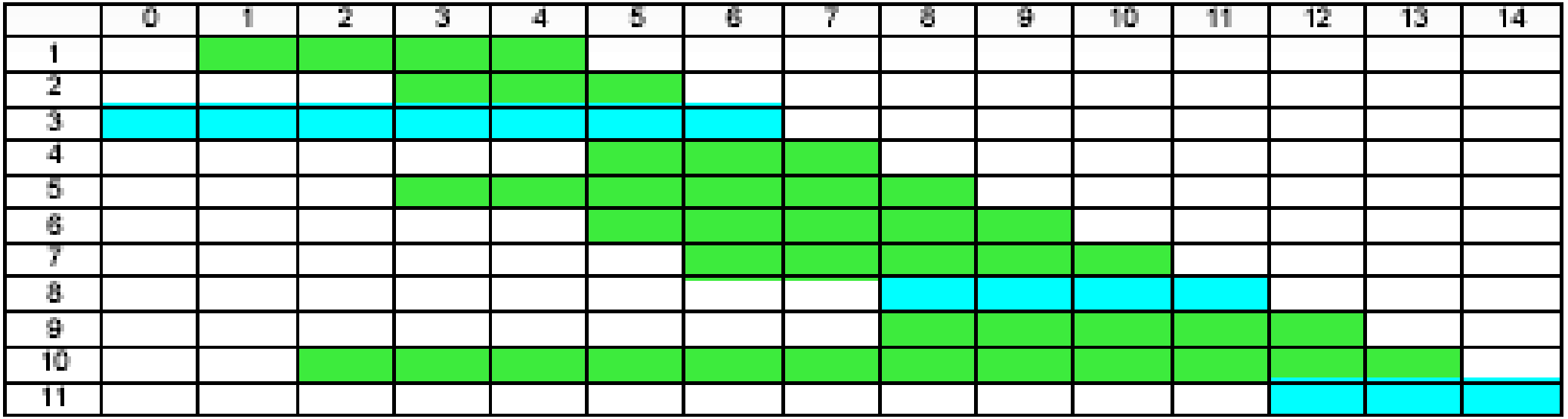
  - Primeira tentativa:
    - começar pelas atividades que começam primeiro
    - Solução foi boa?
      - escolheu 3 atividades: 3, 8 e 11
      - quantas poderiam ter sido escolhidas?



Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

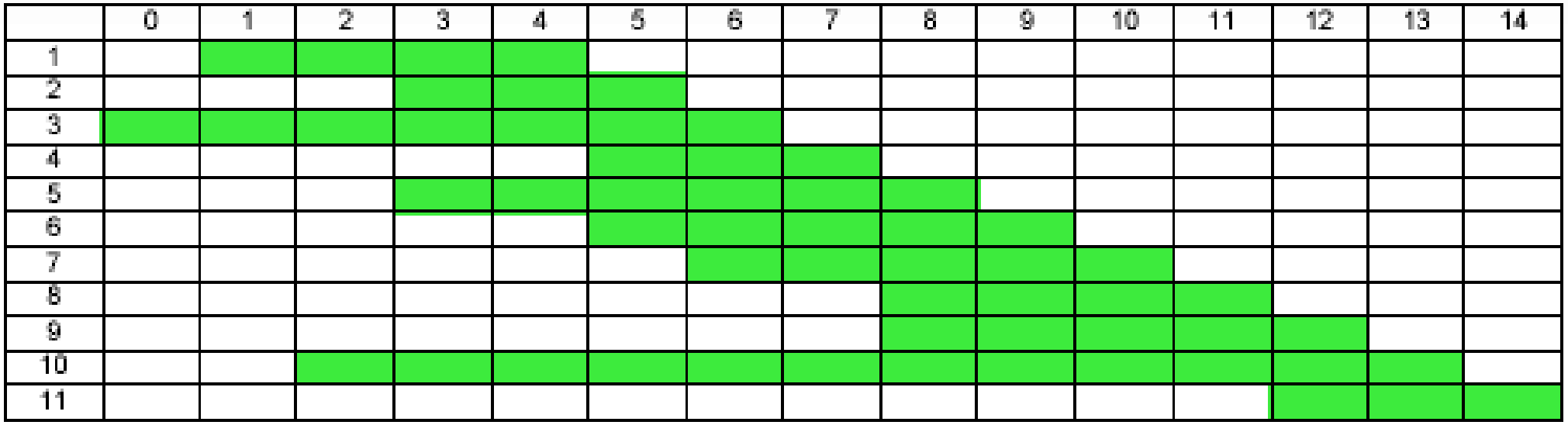
  - Primeira tentativa:
    - começar pelas atividades que começam primeiro
    - Solução foi boa?
      - escolheu 3 atividades: 3, 8 e 11
      - quantas poderiam ter sido escolhidas?
        - 4 atividades: 1, 4, 8 e 11



Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

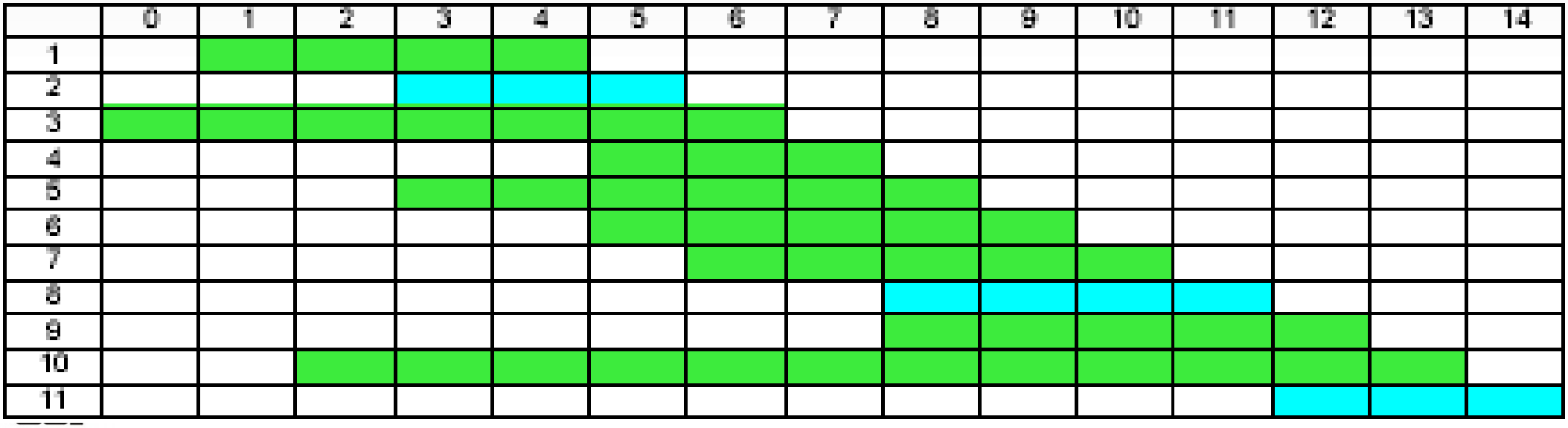
  - Segunda tentativa:
    - começar pelas atividades que duram menos tempo
    - Como fica?



Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

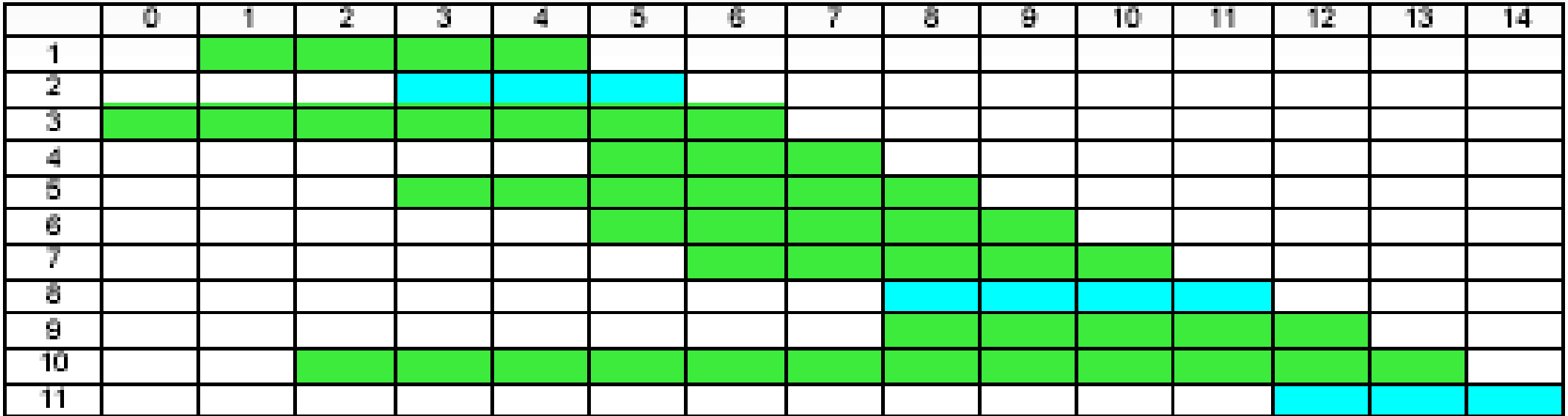
  - Segunda tentativa:
    - começar pelas atividades que duram menos tempo
    - Como fica?



Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

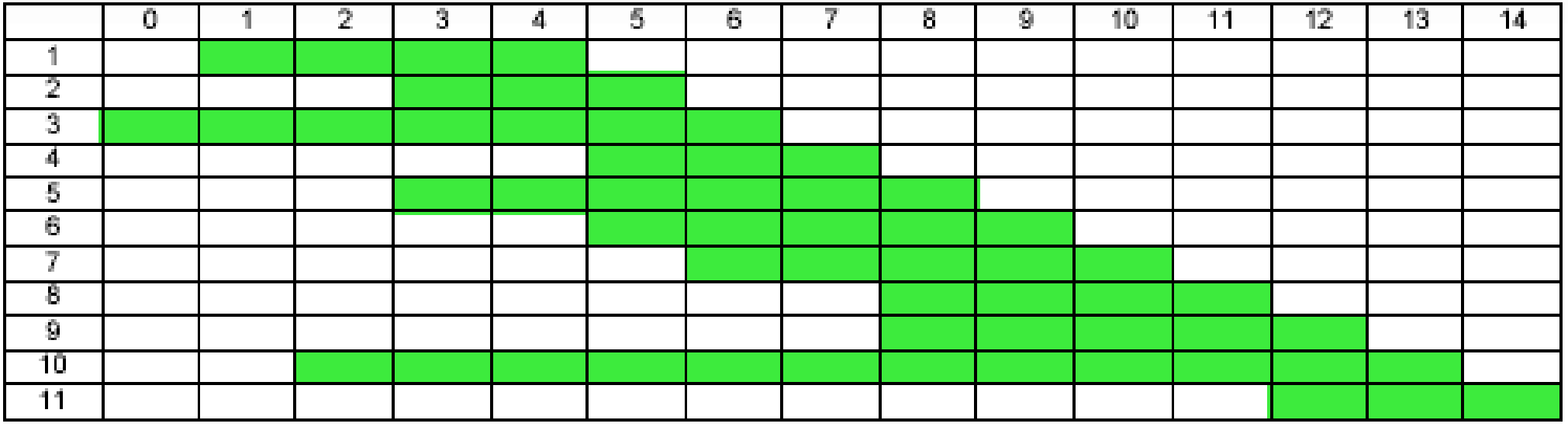
  - Segunda tentativa:
    - começar pelas atividades que duram menos tempo
    - Solução foi boa?
      - escolheu 3 atividades: 2, 8 e 11
      - quantas poderiam ter sido escolhidas?
        - 4 atividades: 1, 4, 8 e 11



Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

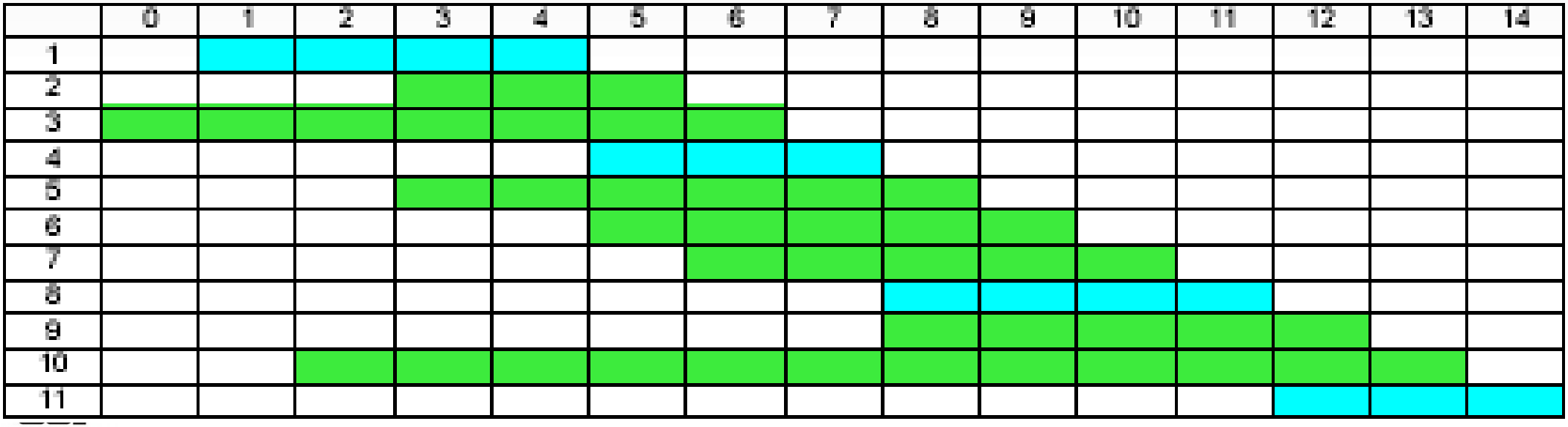
  - Terceira tentativa:
    - começar pelas atividades que terminam primeiro
    - Como fica?



Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

  - Terceira tentativa:
    - começar pelas atividades que terminam primeiro
    - Como fica?
    - Solução foi boa?
      - escolheu 4 atividades!



Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

  - Algoritmo para o problema de locação de atividades

Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

  - Algoritmo para o problema de locação de atividades

    - recebe a lista de atividades ordenadas pelo horário de término;
    - a cada iteração verifica se a atividade atual pode ser incluída na lista de atividades;
    - atividade atual é a que termina primeiro, dentre as atividades restantes.

Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

```
...
void main()
{
  // as atividades devem ser ordenadas pelo campo fim
  // ou seja, as atividades que acabam primeiro ficam na frente
  int[] inicio = {1, 3, 0, 5, 3, 5,  6,  8,  8,  2, 12};
  int[] fim    = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14};
  int numeroDeAtividades = 11;

  int total = selecaoGulosa(inicio, fim, numeroDeAtividades);
  printf("Foram selecionadas %d atividades.\n", total);

}
     ...
```

Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

```
/* parâmetros: inicio e fim de n atividades */
/* retorna o nr de ativ selecionadas, e imprime a ordem */
    int selecaoGulosa(int[] ini, int[] fim, int n)
    {
          int ultimaSelecionada = -1;
          int selecionadas = 0;
          if (n <= 0) return 0;
          // a primeira atividade é sempre selecionada
          printf("Ativ 0, ");
          selecionadas++;
        ultimaSelecionada = 0;
          for (int i = 1; i < n; i++)
             if (ini[i] >= fim[ultimaSelecionada])
             {
                    printf("Ativ %d, ", i);
                    selecionadas++;
                    ultimaSelecionada = i;
             }
          printf("\n");
          return selecionadas;
    }


    ...
```

Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

```
/* parâmetros: inicio e fim de n atividades */
/* retorna o nr de ativ selecionadas, e imprime a ordem */
    int selecaoGulosa(int[] ini, int[] fim, int n)
    {
          int ultimaSelecionada = -1;
          int selecionadas = 0;
          if (n <= 0) return 0;
          // a primeira atividade é sempre selecionada
          printf("Ativ 0, ");
          selecionadas++;
        ultimaSelecionada = 0;
          for (int i = 1; i < n; i++)
             if (ini[i] >= fim[ultimaSelecionada])
             {
                    printf("Ativ %d, ", i);
                    selecionadas++;
                    ultimaSelecionada = i;
             }
          printf("\n");
          return selecionadas;
    }
```

Prova de corretude e otimalidade: Teorema 16.1 do livro do Cormen

Complexidade:

...

Algoritmos gulosos

- Exemplo 2: locação de atividades em uma sala

```
/* parâmetros: inicio e fim de n atividades */
/* retorna o nr de ativ selecionadas, e imprime a ordem */
    int selecaoGulosa(int[] ini, int[] fim, int n)
    {
          int ultimaSelecionada = -1;
          int selecionadas = 0;
          if (n <= 0) return 0;
          // a primeira atividade é sempre selec
          printf("Ativ 0, ");
          selecionadas++;
        ultimaSelecionada = 0;
          for (int i = 1; i < n; i++)
              if (ini[i] >= fim[ultimaSelecionada])
              {
                    printf("Ativ %d, ", i);
                    selecionadas++;
                    ultimaSelecionada = i;
              }
          printf("\n");
          return selecionadas;
    }

...
```

```
void main()
{
  // as atividades devem ser ordenadas pelo campo fim
  // ou seja, as atividades que acabam primeiro ficam na frente
  int[] inicio = {1, 3, 0, 5, 3, 5,  6,  8,  8,  2, 12};
  int[] fim    = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14};
  int numeroDeAtividades = 11;

  int total = selecaoGulosa(inicio, fim, numeroDeAtividades);
  printf("Foram selecionadas %d atividades.\n", total);

}
```

Prova de corretude e otimalidade: Teorema 16.1 do livro do Cormen

Complexidade: O(?) + O(n) = O(?) do tempo do alg de ordenação usado

Algoritmos gulosos

- Generalizando e formalizando:
  - Dado um conjunto *C*, deseja-se determinar um subconjunto $S \subseteq C$, tal que:
    - *(i)* *S* satisfaça uma dada propriedade *P; (S é viável)*
    - *(ii)* *S* é mínimo (ou máximo) em relação a algum critério α (*S* é o menor ou maior subconjunto de *C*, segundo α, que satisfaz *P.) (S é ótimo)*
- Para resolver este problema, o algoritmo guloso:
  - executa um processo iterativo em que *S* é construído adicionando-se elementos de *C*, um a um, sem remover o que foi anteriormente adicionado, e sem reconsiderar o que já foi anteriormente descartado.

Algoritmos gulosos

- Como fazer um algoritmo geral?

Algoritmos gulosos

- Como fazer um algoritmo geral?

```
Guloso (conjunto C)
{
 S ← ∅
 enquanto  (C ≠ ∅) e não encontrou solução faça
 {
      x ← selecionaGulosamente(C) //melhor possível
      C ← C – x
      se viável (S + x)
           S ← S + x
 }
 se S é solução
      retorna S
 senão
      imprime (‘Não existe solução’)
}
```

Exercício

(Cormen 16.2-5) Descreva um algoritmo eficiente que dado um conjunto $\{x_1, \ldots, x_n\}$ de pontos na reta real, determina a menor coleção de intervalos fechados unitários (tamanho 1) que contém todos os dados pontos (menor coleção significa menor número de intervalos). Prove que seu algoritmo está correto. Forneça e justifique a complexidade do seu algoritmo.

Há outros exercícios e notas em https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/guloso.html

Referências

- Nívio Ziviani. Projeto de Algoritmos com implementações em C e Pascal. Editora Thomson, 2a. Edição, 2004, seção 2.7 (texto base)
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. Algoritmos - Tradução da 3a. Edição Americana. Editora Campus, 2012. Cap 16
- Notas de aula – Prof. Norton Roman – EACH-USP
- Notas de aula – Prof. Delano Beder – EACH-USP