Algoritmos Gulosos
Seleção de Atividades

Grupo 02:
Bruno Lírio
Thales Veras
Pedro Paulo

Motivação

O problema da seleção de atividades, dado um conjunto de atividades, definidas por seus tempos de início e fim, constitui-se em alocar o maior número possível de atividades em um período de tempo definido, por exemplo, um conjunto de palestras ou de aulas que devem ser alocadas em um auditório ou em uma sala de aula.

Consiste em construir a solução aos poucos. O algoritmo guloso para este problema escolhe o candidato, e ao ser adicionado à solução, permanece sem possibilidade de ser substituido. Opostamente, uma vez que um candidato é excluído do conjunto solução, ele nunca mais é reconsiderado.

O algoritmo procede acrescentando os novos elementos, um de cada vez.

Exigências para seleção de atividades:

- Existem diversas atividades (aulas) que querem usar um mesmo recurso (uma sala de aula).
- 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, ou seja, conflito de horário.

Definição Formal do Problema (Entrada / Questão / Saída)

- Entrada: Conjunto de atividades, com respectivos tempos de duração.
- Questão: Verificar qual o maior número de atividades podem fazer uso de um recurso sem que haja conflito em relação aos tempos de duração.
- Saída: Após uma seqüência de decisões, o maior número de atividades é selecionado.

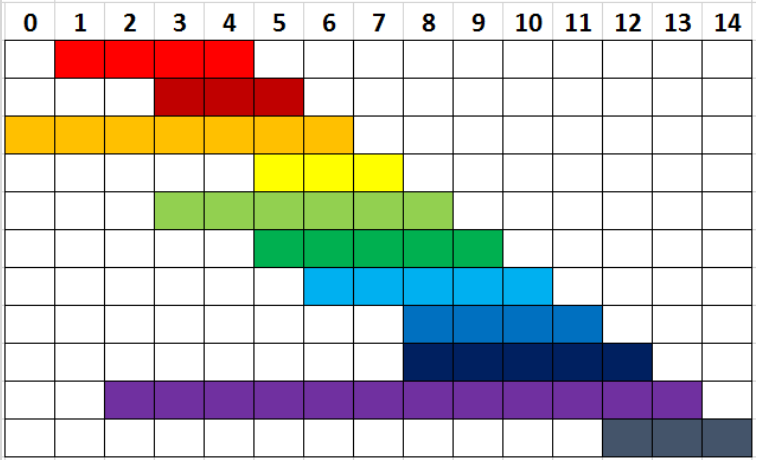
Obs.:Na seqüência de decisões, nenhum elemento é examinado mais de uma vez: ou ele fará parte da saída, ou será descartado.

ncias do Problema

Conjunto de Atividades

i	1	2	3	4	5	6	7	8	9	10	11
s[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

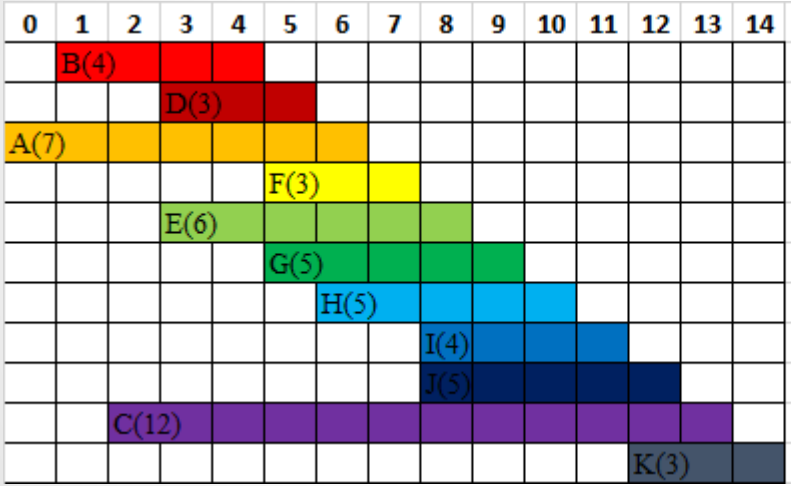
Subconjunto de atividades mutuamente compatíveis máximo



Dado um conjunto S com n atividades, onde s_i e f_i são respectivamente o tempo de início e de fim da atividade a, 1 <= i <= n, onde n são aulas. Duas atividades são compatíveis entre si quando seus tempos de início e de fim não se sobrepõem. O seu objetivo é encontrar um subconjunto máximo de A com atividades mutuamente compatíveis.

Descrição ideia do Algoritmo

Inicialmente as atividades estão ordenadas de forma crescente pelo tempo de finalização da atividade.



A(0,6)		B(1,4)
B(1,4)		D(3,5)
C(2,13)		A(0,6)
D(3,5)		F(5,7)
E(3,8)		E(3,8)
F(5,7)	→	G(5,9)
G(5,9)		H(6,10)
H(6,10)		I(8,11)
I(8,11)		J(8,12)
J(8,12)		C(2,13)
K(12,14)		K(12,14)

Dada uma coleção de atividades, digamos S, dadas em intervalos, determinar um subconjunto sem sobreposição (compatíveis) máximo de atividades de S, por exemplo da imagem acima, são 11 atividades, em 14 unidades de tempo.

Descrição da ideia do Algoritmo

A ideia do algoritmo consiste em que dado o conjunto de atividades S, extrair de S um subconjunto S*ij* = {a_k ∈ S |f_i ≤ s_k ≤ f_k ≤ s_j} a atividade com tempo de término mais antigo. Suponhamos que seja uma atividade a_m, temos então Sim e Smj, e analisamos agora S*mj* já S*im* é vazio para nossa busca pois, se em S*im* houvesse uma atividade com tempo de término menor que a_m começaríamos por esta atividade. Supondo agora em S*mj* a atividade com tempo de término mais antigo a_w temos S*mw* e S*wj* e assim sucessivamente, até obtermos o subconjunto desejado.

Solução para a Instância

O problema de Seleção de atividades consiste agendar a utilização de um determinado recurso por um conjunto de atividades. Suponhamos que tenha um conjunto S = {a1, a2, .., an} de n atividades, onde a_1, a_2,...,a_n são cada aula, as propostas que desejam utilizar um mesmo recurso, recurso tal como uma sala de aula, que pode ser utilizada por apenas uma atividade por um determinado período de tempo. Cada atividade tem um tempo de início s_i e um tempo de término f_i, onde si < fi. Duas atividades a_i e a_j são compatíveis se os intervalos [s_i, f_i) e [s_j , f_j) não se sobrepõem, i.e., se s_i >= f_j ou se s_j >= f_i. O problema de Seleção de Atividades consiste em encontrar o subconjunto de tamanho máximo de atividades mutuamente compatíveis.

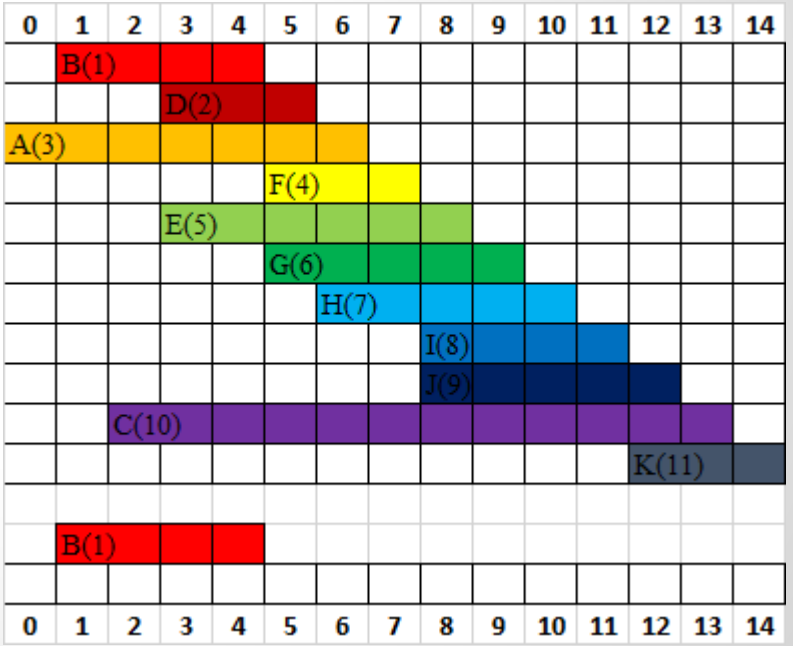
Descrição Formal do Algoritmo

```
SeleçãoAtividades(s, f)
n ← length(s)
S ←{1}
j ←1
for i ←2 to n do
    if s_i >= f_j then {
        S ← S ∪{i}
        j ← i
    }
Return S
```

Etapas da Aplicação da Técnica (solução da instância)

```
SeleçãoAtividades(s, f)
n ← length(s)
S ←{1}
j ←1
for i ←2 to n do
    if s_i >= f_j then {
        S ← S ∪{i}
        j ← i
    }
Return S
```

Primeira Etapa:
Ínicio: conjunto solução S recebe primeira aula, aula 1 (B)

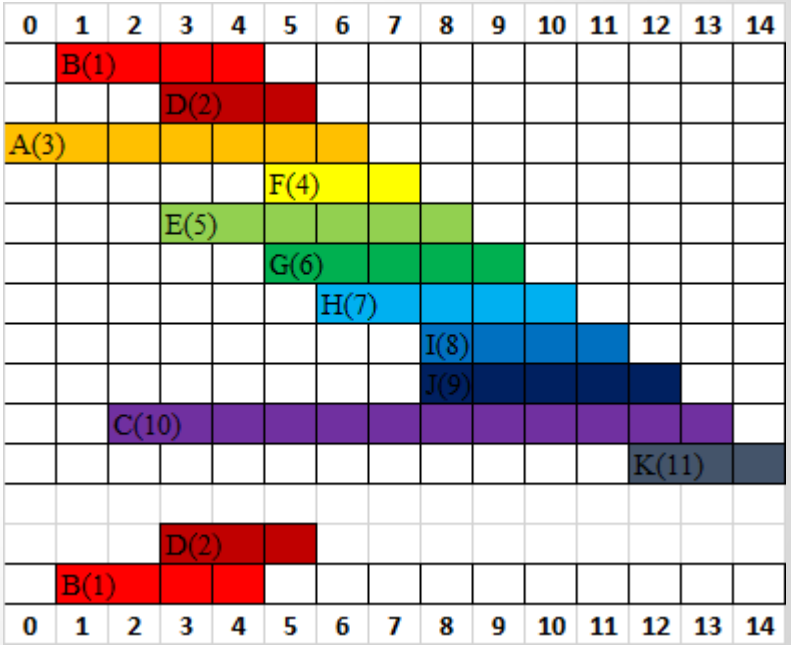


Etapas da Aplicação da Técnica

```
SeleçãoAtividades(s, f)
n ← length(s)
S ←{1}
j ←1
for i ←2 to n do
    if s_i >= f_j then {
        S ← S ∪{i}
        j ← i
    }
Return S
```

Segunda Etapa:

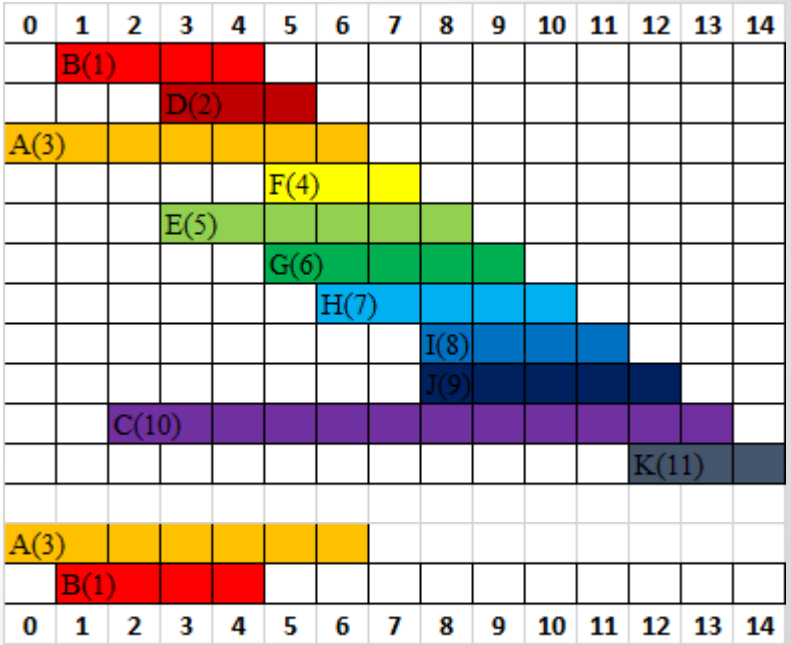
Primeira iteração: Testa se início de aula 2 (D) >= final de aula 1 (B)



Etapas da Aplicação da Técnica

```
SeleçãoAtividades(s, f)
n ← length(s)
S ←{1}
j ←1
for i ←2 to n do
    if s_i >= f_j then {
        S ← S ∪{i}
        j ← i
    }
Return S
```

Terceira Etapa:
Testa início de A >= final de B



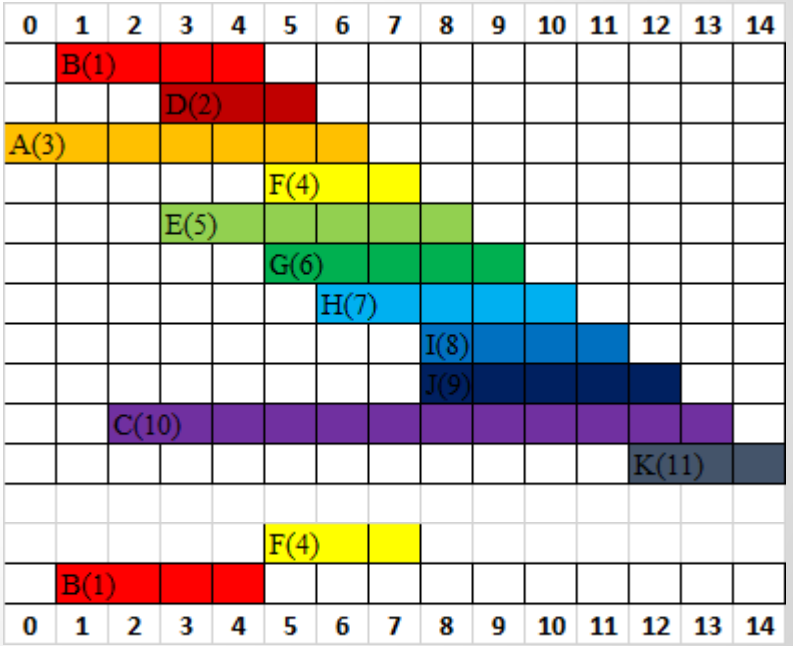
Etapas da Aplicação da Técnica

```
SeleçãoAtividades(s, f)
n ← length(s)
S ←{1}
j ←1
for i ←2 to n do
    if s_i >= f_j then {
        S ← S ∪{i}
        j ← i
    }
Return S
```

Quarta Etapa:

Testa início F >= final de B.

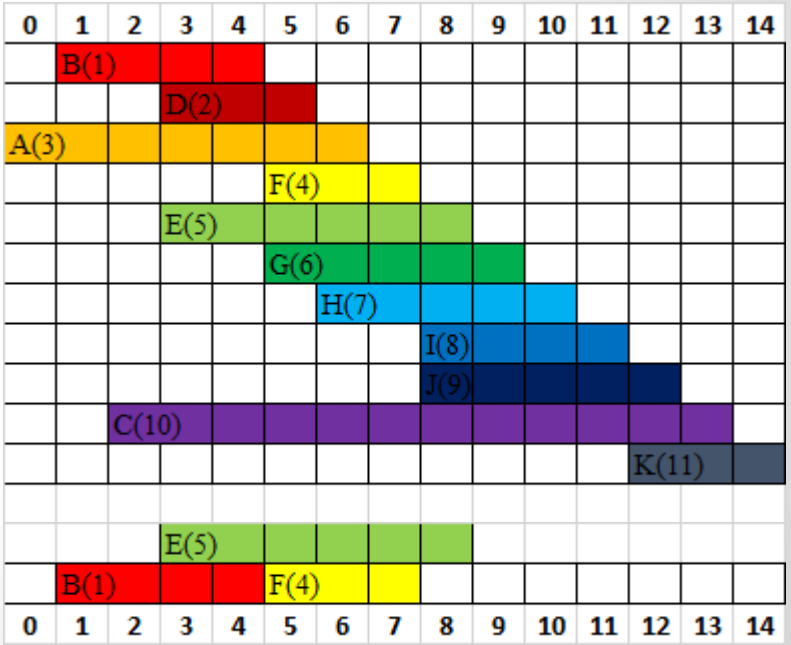
Ok: aula 4 (F) adiciona ao conjunto solução S



Etapas da Aplicação da Técnica

```
SeleçãoAtividades(s, f)
n ← length(s)
S ←{1}
j ←1
for i ←2 to n do
    if s_i >= f_j then {
        S ← S ∪{i}
        j ← i
    }
Return S
```

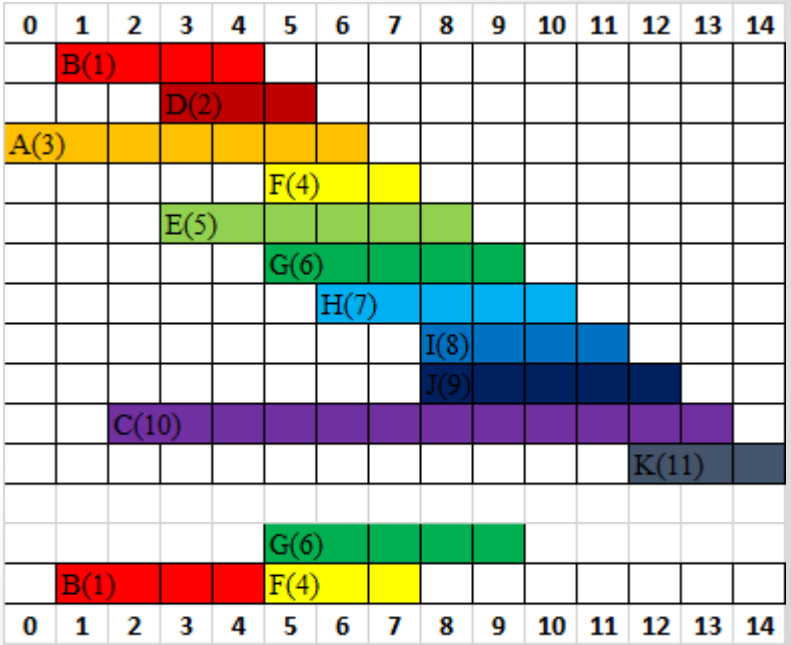
Quinta Etapa:
Testa início de E >= final de F



Etapas da Aplicação da Técnica

```
SeleçãoAtividades(s, f)
n ← length(s)
S ←{1}
j ←1
for i ←2 to n do
    if s_i >= f_j then {
        S ← S ∪{i}
        j ← i
    }
Return S
```

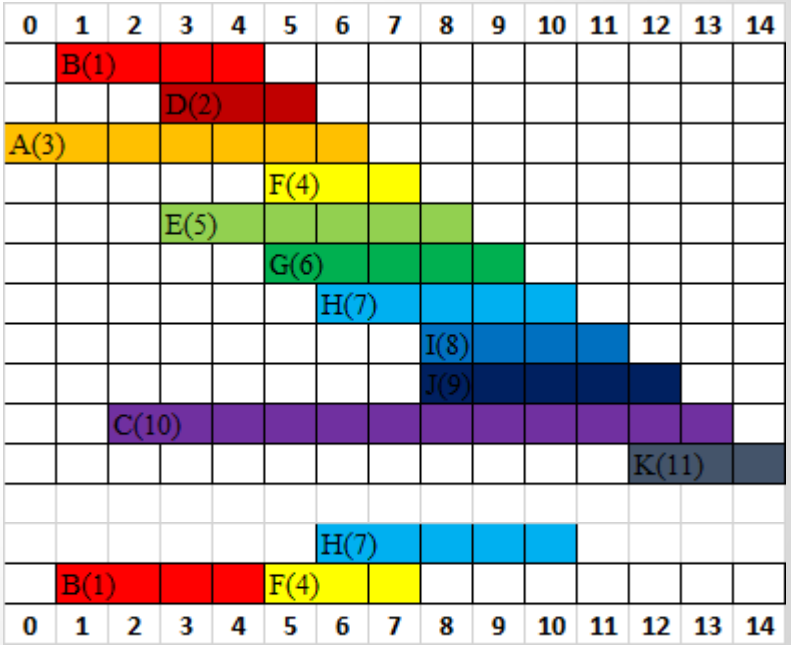
Sexta Etapa:
Testa início de G >= final de F



Etapas da Aplicação da Técnica

```
SeleçãoAtividades(s, f)
n ← length(s)
S ←{1}
j ←1
for i ←2 to n do
    if s_i >= f_j then {
        S ← S ∪{i}
        j ← i
    }
Return S
```

Sétima Etapa:
Testa início de H >= final de F



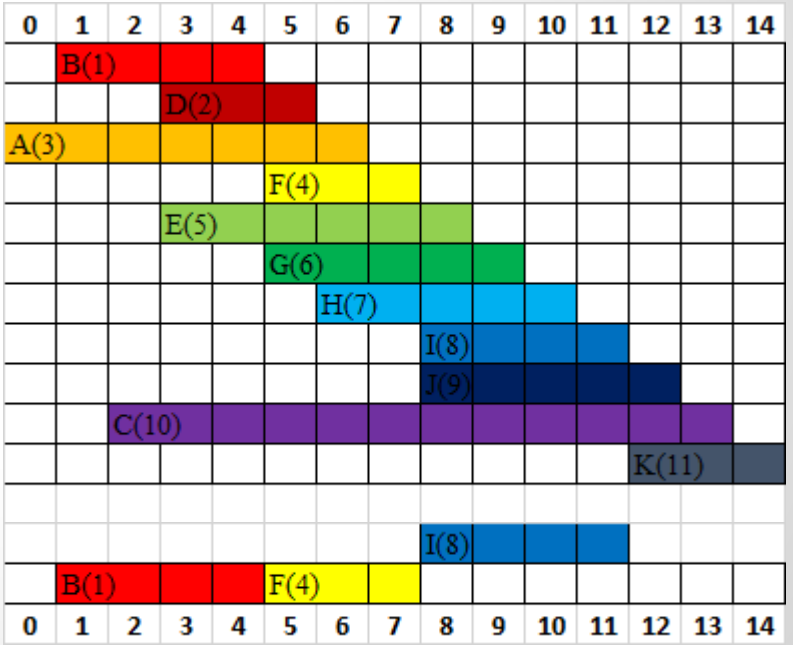
Etapas da Aplicação da Técnica

```
SeleçãoAtividades(s, f)
n ← length(s)
S ←{1}
j ←1
for i ←2 to n do
    if s_i >= f_j then {
        S ← S ∪{i}
        j ← i
    }
Return S
```

Oitava Etapa:

Testa início I >= final de F.

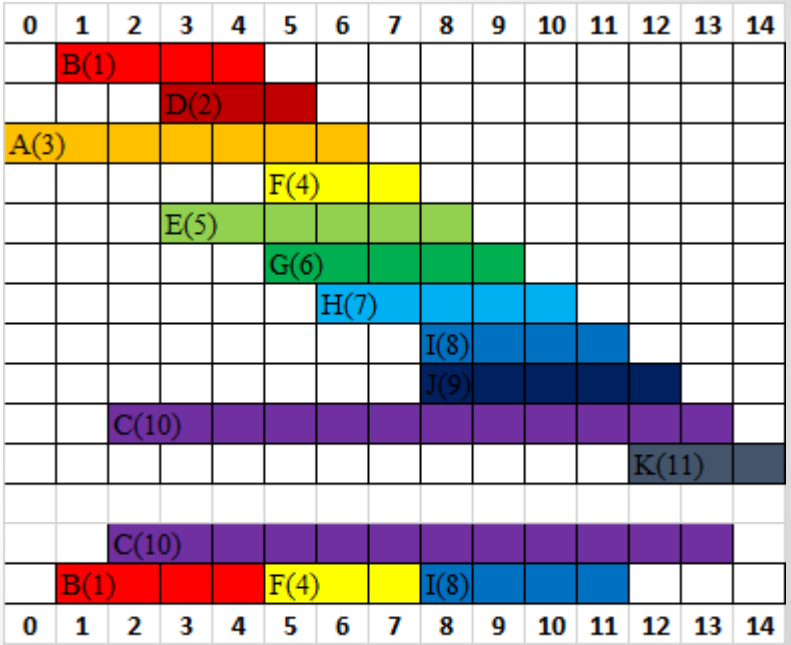
Ok: Adiciona I ao conjunto solução S.



Etapas da Aplicação da Técnica

```
SeleçãoAtividades(s, f)
n ← length(s)
S ←{1}
j ←1
for i ←2 to n do
    if s_i >= f_j then {
        S ← S ∪{i}
        j ← i
    }
Return S
```

Nona Etapa:
Testa início C >= final de I.



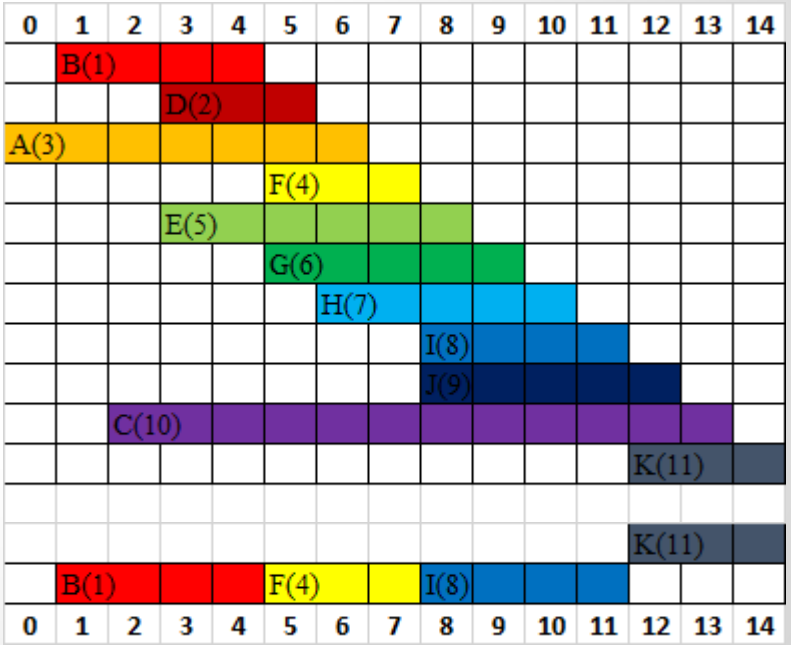
Etapas da Aplicação da Técnica

```
SeleçãoAtividades(s, f)
n ← length(s)
S ←{1}
j ←1
for i ←2 to n do
    if s_i >= f_j then {
        S ← S ∪{i}
        j ← i
    }
Return S
```

Décima Etapa:

Testa início K >= final de I.

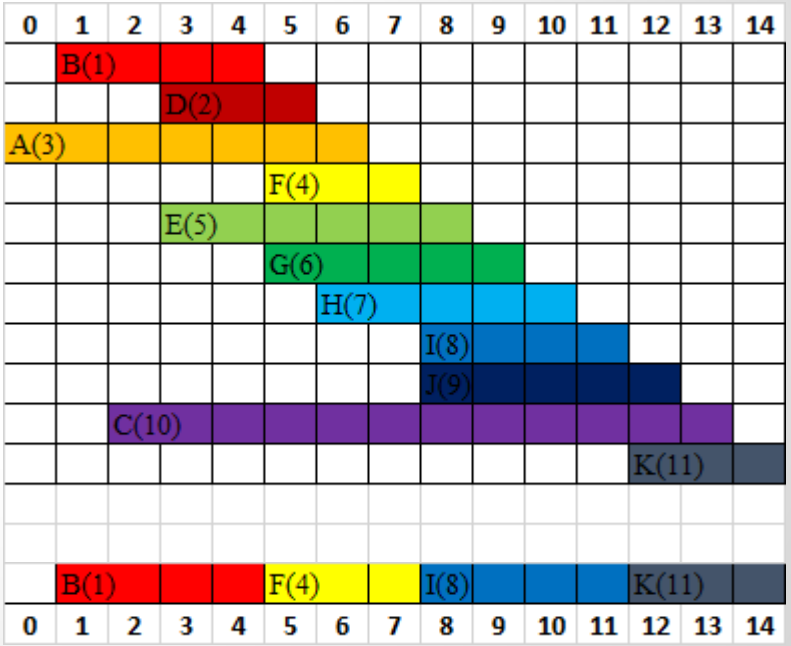
Ok: Adiciona K ao conjunto solução S.



Etapas da Aplicação da Técnica

```
SeleçãoAtividades(s, f)
n ← length(s)
S ←{1}
j ←1
for i ←2 to n do
    if s_i >= f_j then {
        S ← S ∪{i}
        j ← i
    }
Return S
```

Conjunto solução:
S = {B,F,I,K}



Complexidade

- A ordenação é feita em O(n log n).
- O loop é realizado n-1 vezes nos dando a complexidade Θ(n).
- Portanto, o algoritmo é afetado principalmente pela ordenação, e sua complexidade é O(n log n).

Corretude

Consideremos a seguinte implementação:

```
SeleçãoAtividades(s, f)
n ← length(s)
A ←{1}
j ←1
for i ←2 to n do
      if s_i >= f_j then {
            A ← A ∪{i}
            j ← i
      }
Return A
```

Corretude

Supondo os tempos de término das atividades ordenados monotonicamente e S={1,2,3,...,n}, então temos:

$$f_1 \leq f_2 \leq \ldots \leq f_n$$

Assim f_1 é o tempo de término mais antigo. Vamos mostrar que existe uma solução ótima A tal que $1 \in A$. Seja $C \subseteq S$ uma solução ótima, vamos supor que as atividades também estão ordenadas em C. Seja k a 1ª atividade escolhida em C. Então necessariamente $f_1 \leq f_k$. Portanto a solução $A = C - \{k\} \cup \{1\}$ é viável e é maximal, pois $|A| = |C|$ (com isso queremos dizer que os conjuntos vão diferir apenas pela diferença de seus elementos mas, o número deles continua o mesmo).

Uma vez que A é uma solução ótima para S, então $A1 = A - \{1\}$ é uma solução ótima para o problema gerado por $S1 = \{i \in S \mid s_i \geq f_1\}$ (ou seja, atividades que possuem início após o fim de 1). Se fosse possível conseguir uma solução B1 com mais atividades que A1, então seria possível conseguir uma solução B para S com mais atividades que A o que seria uma contradição.

Então, após a escolha local ótima, resta um subproblema dado por $S1 = \{i \in S \mid s_i \geq t_1\}$, independente da primeira escolha, que deve conter em sua solução a atividade com o menor tempo de término. Por fim, esta mecânica de escolha se propaga ao longo das decisões.

Conclusão e Discusões

Vantagens:

- Algoritmo simples.
- Fácil implementação.
- Geralmente é fácil de construir um algoritmo.
- Baixa complexidade.

Desvantagens:

- Não há nenhuma garantia de que obteremos os outros conjuntos maximais ou se ao menos saberemos de sua existência.

Conclusão e Discusões

Podemos concluir que o algoritmo sempre nos fornecerá um conjunto maximal mas, que não necessariamente seja o único.

Possível ter um algoritmo melhor?

Somente se considerarmos que o conjunto de atividades está previamente ordenado em relação aos seus respectivos tempos de término.

Refêrencias Bibliograficas

Algoritmos gulosos: definições e aplicações (2004). UNICAMP. ***Site.*** Disponível em: <http://www.ic.unicamp.br/~rocha/msc/complex/algoritmosGulososFinal.pdf>. Acesso em: 30 out. 2013

Activity selection problem (2013). WIKIPEDIA. ***Site.*** Disponível em: <http://en.wikipedia.org/wiki/Activity_selection_problem>. Acesso em: 30 out. 2013

Greedy Algorithms Activity selection problem. GEEKSFORGEEKS. ***Site.*** Disponível em: <http://www.geeksforgeeks.org/greedy-algorithms-set-1-activity-selection-problem/>. Acesso em: 30 out. 2013

Cormen, Algoritmos Teoria e Prática 2ª ed. ELSEVIER.