Algoritmos Avançados

Busca Binária

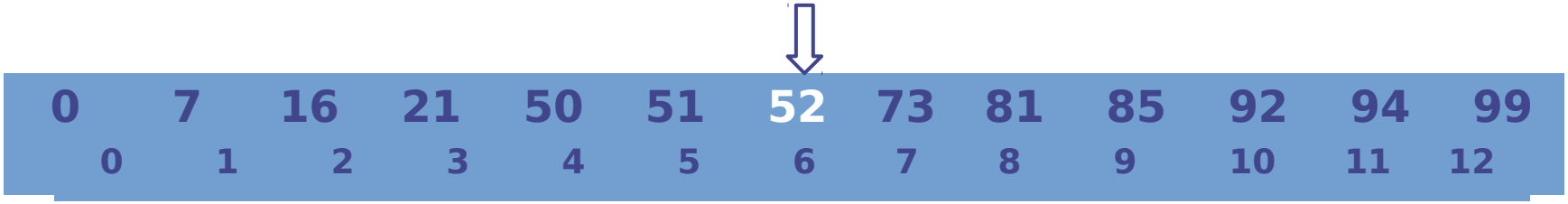
João Batista

Busca Binária

- Busca binária possui algumas aplicações pouco conhecidas.
- A aplicação mais conhecida envolve um vetor, com as seguintes restrições:
  - Existe uma função de comparação de ordem entre os elementos;
  - Os elementos do vetor estão ordenados segundo essa função de comparação.

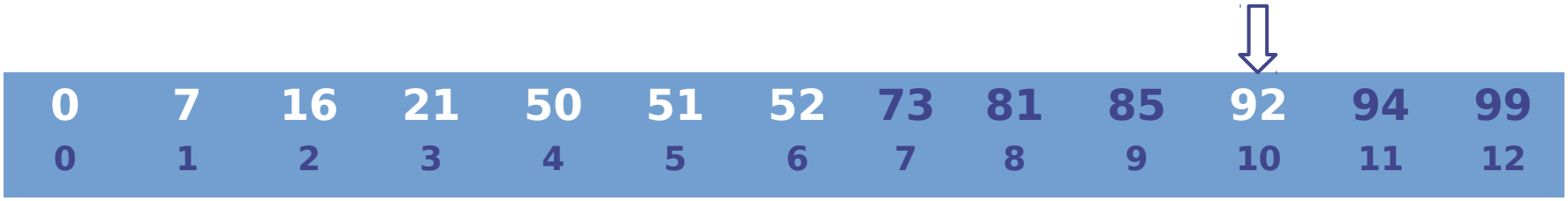
Busca Binária em Vetores

- Por exemplo, considere o vetor abaixo, para uma chave de busca = 73.
- O algoritmo realiza três passos:
  - Verifica o valor do elemento central, e para se igual a chave procurada;
  - Se o valor central for menor do que a chave, a metade superior é examinada na próxima iteração;
  - Se o valor central for maior do que a chave, a metade inferior é examinada.



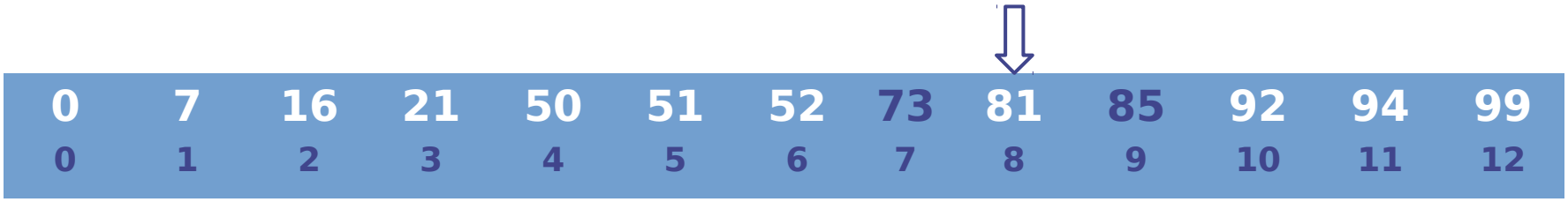
Busca Binária em Vetores

- Por exemplo, considere o vetor abaixo, para uma chave de busca = 73.
- O algoritmo realiza três passos:
  - Verifica o valor do elemento central, e para se igual a chave procurada;
  - Se o valor central for menor do que a chave, a metade superior é examinada na próxima iteração;
  - Se o valor central for maior do que a chave, a metade inferior é examinada.



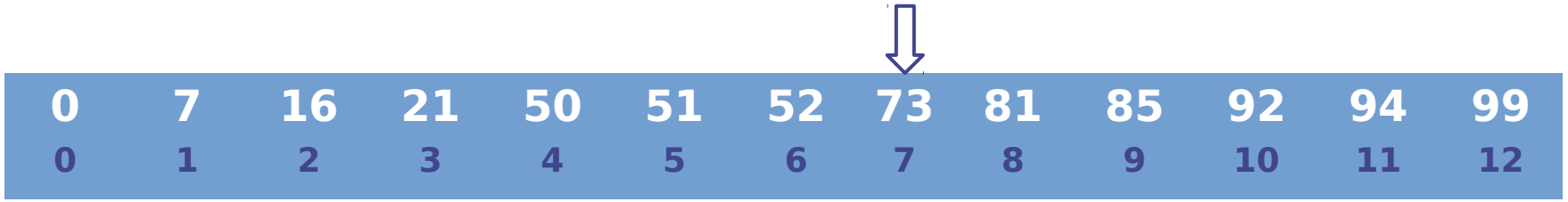
Busca Binária em Vetores

- Por exemplo, considere o vetor abaixo, para uma chave de busca = 73.
- O algoritmo realiza três passos:
  - Verifica o valor do elemento central, e para se igual a chave procurada;
  - Se o valor central for menor do que a chave, a metade superior é examinada na próxima iteração;
  - Se o valor central for maior do que a chave, a metade inferior é examinada.



Busca Binária em Vetores

- Por exemplo, considere o vetor abaixo, para uma chave de busca = 73.
- O algoritmo realiza três passos:
  - Verifica o valor do elemento central, e para se igual a chave procurada;
  - Se o valor central for menor do que a chave, a metade superior é examinada na próxima iteração;
  - Se o valor central for maior do que a chave, a metade inferior é examinada.



Busca Binária em Vetores

- Como cada iteração do algoritmo, descarta-se metade do vetor:
  - Complexidade: $O(\log n)$.
- Existem implementações prontas de busca binária:
  - C: bsearch
  - STL: (bool) binary_search, (iterator) lower_bound, (iterator) upper_bound
  - As implementações STL são $O(\log n)$ para iteradores aleatórios e $O(n)$ para sequenciais.

Busca Binária: Algoritmo

```
int binary_search(int A[], int key, int imin, int imax)
{
  // test if array is empty
  if (imax < imin)
    // set is empty, so return value showing not found
    return KEY_NOT_FOUND;
  else
    {
      // calculate midpoint to cut set in half
      // watch for overflow!
      int imid = imin + ((imax - imin) / 2);

      // three-way comparison
      if (A[imid] > key)
        // key is in lower subset
        return binary_search(A, key, imin, imid - 1);
      else if (A[imid] < key)
        // key is in upper subset
        return binary_search(A, key, imid + 1, imax);
      else
        // key has been found
        return imid;
    }
}
```

Método da Bissecção

- Suponha que você precisa pagar um empréstimo com juros. Por exemplo, um empréstimo de $1000, em dois meses, com juros de 10%.
- Qual valor deve ser pago para que, ao final de dois meses, o saldo seja zero?
  - $1000 x 1.1 – $576.19 = $523.81
  - $523.81 x 1.1 – $576.19 = $0
- Como obter o valor de $576.19?

Método da Bissecção

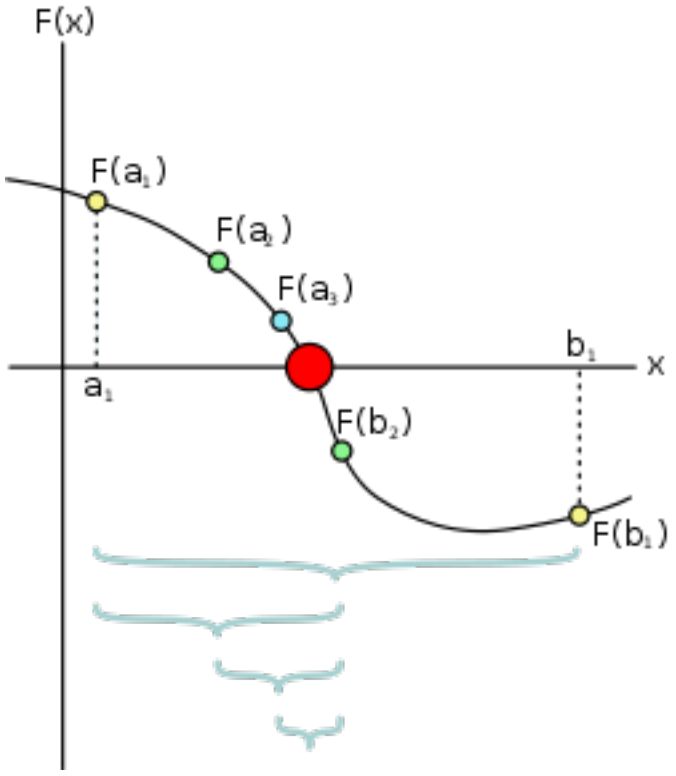
a	***b***	***d* = (*a*+*b*) / 2**	***f* (1000,2,10%,*d*)**	**ação**
0.01	1100.00	550.005	54.9895	aumentar
550.005	1100.00	825.0025	-522.50525	diminuir
550.005	825.0025	687.50375	-233.757875	diminuir
...	...	...	...	...
...	...	576.190476	|error| < ε	parar

$f\,(1000,2,10\%,d) = 1000 \times 1.1 - d + (1000 \times 1.1 - d) \times 1.1 - d$

Complexidade: $O(\log_2(b-a)/\varepsilon)$

Método da Bissecção

- É preciso estimar um intervalo [*a*, *b*] onde a raiz se encontra
- *f* (*a*) e *f* (*b*) devem ter sinais opostos.
- *f* deve ser contínua.



Método da Bissecção

```
double bisection(double lo, double hi){
  while (hi - lo > eps){
    double center = (lo + hi) / 2;
    if (f(lo) * f(center) <= 0){
      hi = center;
    } else {
      lo = center;
    }
  }
  return (lo+hi)/2;
}
```

Busca Binária, a solução

- Você quer cruzar o deserto. Seu jeep possui um tanque “grande o bastante”, inicialmente cheio. Durante a viagem, você pode encontrar os seguintes eventos:
  - Dirigir (consome combustível)
  - Vazamento (reduz ainda mais o combustível)
  - Posto (enche novamente o tanque)
  - Mecânico (conserta todos os vazamentos)
  - Atingiu o objetivo (termina a viagem !).
- >> Qual o menor tanque para que você consiga terminar a viagem ????

Busca Binária, a solução

- Se a capacidade do tanque for dada, o problema é de simulação, mas esse valor não é dado e deve ser determinado na faixa de [0.000 .. 10000.000]
- Esse problema tem uma característica que pode ser explorada. Se a resposta é *x, então*:
  - [0.000 .. *x* – 0.001] não leva o jeep ao destino. Mas,
  - [*x* .. 10000.000] leva o jeep ao destino, possivelmente com algum combustível sobrando no tanque.
- Portanto, podemos usar essa característica para buscar pela resposta.

Busca Binária, a solução

- Segundo [1], um problema aceita uma busca binária na resposta, se:
  - É um problema de otimização do tipo “ache o menor (maior) *X* tal que tais condições sejam verdadeiras”?
  - O problema possui a propriedade de que se para um *X* as condições são verdadeiras então para todos os valores maiores (menores) que *X* as condições tambem são verdadeiras?
  - Dado um valor *X*, é possível verificar de forma rápida se as condições são verdadeiras para esse *X*?

[1] http://community.topcoder.com/tc?module=Static&d1=match_editorials&d2=tco07_rd1a

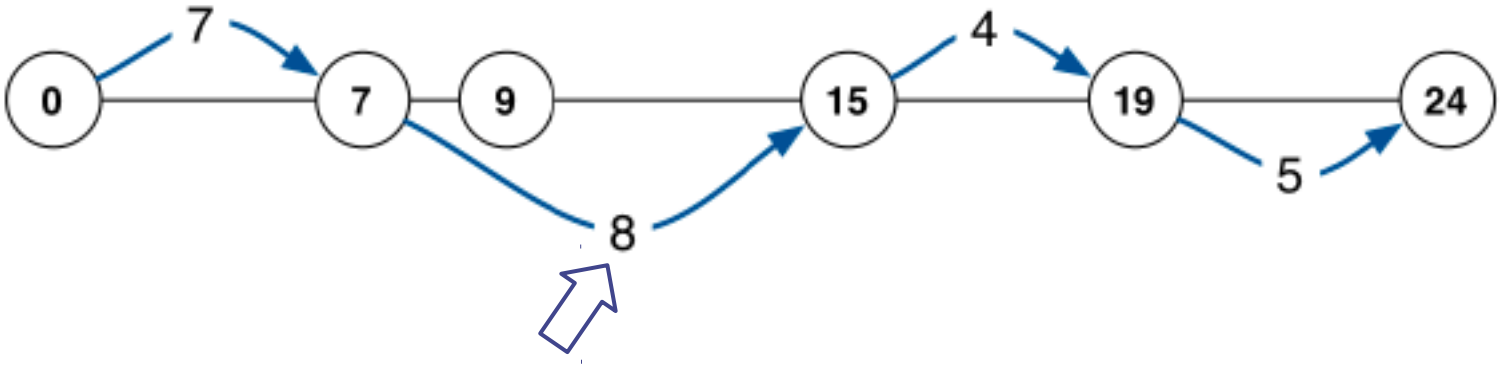
UVa 907 – Winterim Backpacking Trip

This Winter we are going on a trip along the Appalachian Trail. The trail is a continuous marked footpath that goes from Katahdin in Maine to Springer Mountain in Georgia, a distance of about 2160 miles. Even though our trip will only consider some part of the trail, it will be our first real backpacking experience and an excellent opportunity to acquire winter camping skills.

Part of the experience is also the route planning of the trip. We have a list of all possible campsites ($N \leq 600$) that we can use along the way and we want to do this trip so that we only stop K nights to camp ($K \leq 300$). We also know in advance the distance between consecutive campsites and we are only allowed to camp at a campsite. Our goal is to plan the trip so that we minimise the maximum amount of walking done in a single day. In other words, if our trip involves 2 nights (3 days of walking), and we walk 9, 10, 5 miles on each day respectively, the cost (maximum amount of walking done in one day) is 10. Another schedule that involves walking 9, 6, 9 miles on each day has cost 9.

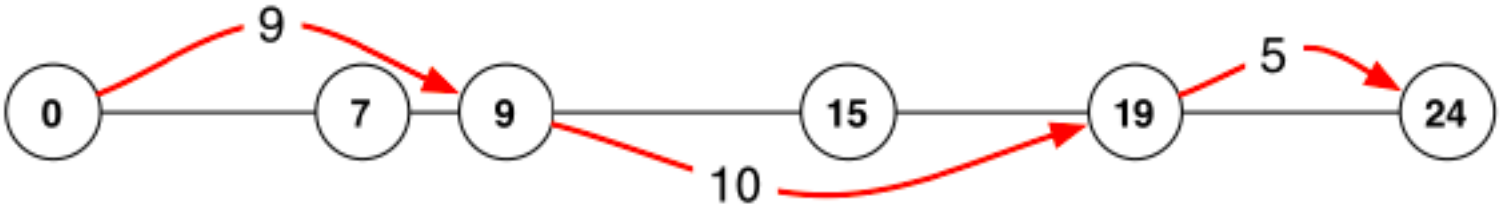
UVa 907 – Winterim Backpacking Trip

- Por exemplo, para $K = 3$ e os acampamentos a seguir, a resposta é 8.



UVa 907 – Winterim Backpacking Trip

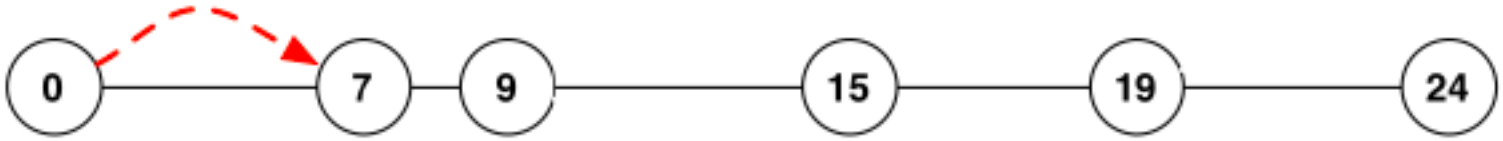
- Iremos fazer uma busca binária no tamanho do passo *P*. *P* pode assumir valores entre 0 e a maior distância, que neste caso é 24.
- Sendo assim, a primeira inspeção é para *P* = 12.



- Para *P* = 12 é possível chegar ao destino com 2 paradas, logo, o próximo intervalo passa a ser [0, 11].

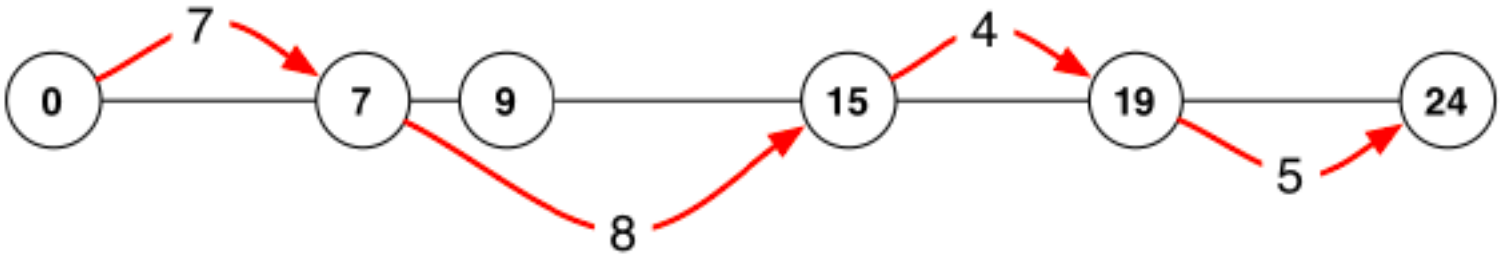
UVa 907 – Winterim Backpacking Trip

- Para o intervalo de [0, 11], o elemento central é 5. Como com 5 não e possível chegar ao fim, o intervalo é atualizado para [6, 11].



UVa 907 – Winterim Backpacking Trip

- Para o intervalo [6, 11], o elemento central é 8. Com 8 e possível chegar ao fim em 3 paradas, logo, o intervalo é atualizado para [6, 7].



- Nas iterações seguintes todos os elementos inspecionados falham. Sendo assim, a nossa resposta é 8 (o último elemento que passou pela vericação).

Conclusão

- Busca binária pode ser utilizada tanto para buscar valores em vetores, quanto raízes de funções e respostas de problemas.
- Os problemas desta aula ilustram essas utilidades:
  - Uva 907 – Winterim Backpacking Trip
  - UVa 10341 – Solve It
  - UVa 11935 – Through the Desert