M1103 Amphi 6 : Recherche

Thomas Nowak

Université Paris-Sud

La dernière fois

Recherche

Recherche linéaire

```
bool recherche(const vector<int> &vec, int x)
{
    for(int i = 0; i < vec.size(); ++i)
        if(vec[i] == x)
            return true;

    return false;
}
```

Recherche linéaire

- complexité en temps $O(\texttt{vec.size()})$
- le nombre d'itérations précis dépend du vecteur `vec` et de l'entier `x`
- pire cas = `x` n'est pas dans `vec`
- nécessite `vec.size()` comparaisons
- ne peut être amélioré en général

Recherche linéaire avec vecteur trié

- sous certains hypothèses sur le contenu du vecteur `vec`, on peut réduire le nombre d'itérations nécessaire
- c'est le cas si `vec` est *trié*, càd `vec[i]` $\leq$ `vec[i+1]`
- propriété clé pour la recherche d'un élément `x` :
  `x` $<$ `vec[i]` $\implies$ `x` $<$ `vec[i+1]`
- permet de retourner `false` dès que `x` $<$ `vec[i]` dans *une* itération

Recherche linéaire avec vecteur trié

```
bool recherche2(const vector<int> &vec, int x)
{
    for(int i = 0; i < vec.size(); ++i)
    {
        if(vec[i] == x)
            return true;
        if(x < vec[i])
            return false
    }

    return false;
}
```

Recherche linéaire avec vecteur trié

- en particulier, si `x` est strictement plus petit que le premier élément (et donc tous les éléments) : une seule itération
- nouveau pire cas : `x` strictement plus grand que tous les éléments
- nécessite `vec.size()` itérations

Recherche dichotomique

- idée : si $x < vec[i]$ alors `x` ne se trouve pas à *droite* de l'indice `i`
- permet d'*éliminer* toute une partie du vecteur pour la recherche de `x`
- pareil : si $x > vec[i]$ alors `x` ne se trouve pas à *gauche* de l'indice `i`
- quel est le meilleur endroit pour essayer de couper le vecteur ?

Recherche dichotomique

```
bool recherche3(const vector<int> &vec, int x) {
    int debut = 0, fin = vec.size() - 1;
    while(debut <= fin) {
        int mil = (debut + fin)/2;
        if(x == vec.at(mil))
            return true;
        if(x < vec.at(mil))
            fin = mil - 1;
        else // x > vec.at(mil)
            debut = mil + 1;
    }
    return false;
}
```

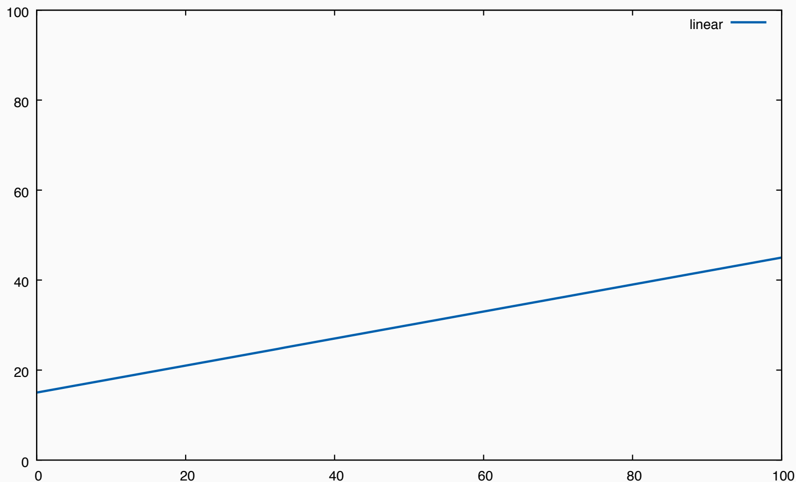
Recherche dichotomique

- on regarde la “fenêtre” entre les indices `debut` et `fin`
- dans chaque itération, soit on trouve l'élément, soit on divise par 2 la taille de la fenêtre
- quel est le nombre d'itérations de la boucle ?
- est-ce que le nombre d'itérations est toujours fini ?

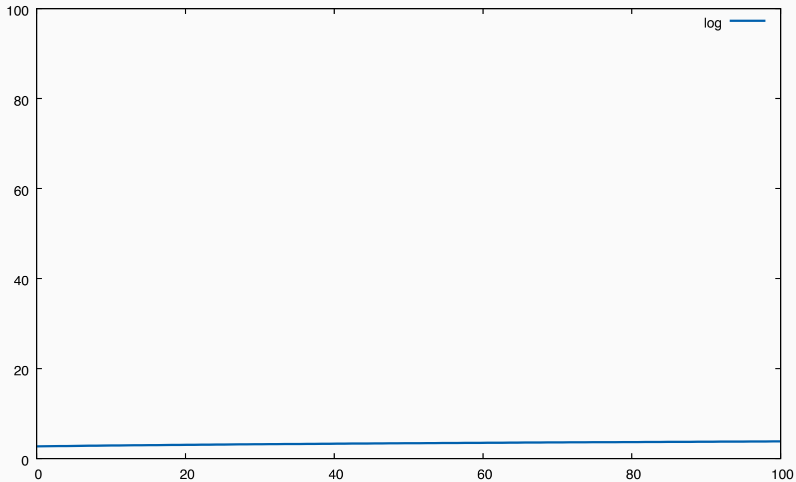
Recherche dichotomique

- le nombre d'itérations est fini : la différence `fin−debut` décrémente dans chaque itération, jusqu'à ce qu'elle devienne négative
- on a même que la différence est divisée au moins par 2
- on a donc besoin de $O\big(\log_2(\texttt{vec.size()})\big)$ itérations
- cela est beaucoup mieux que la complexité linéaire des algorithmes précédents

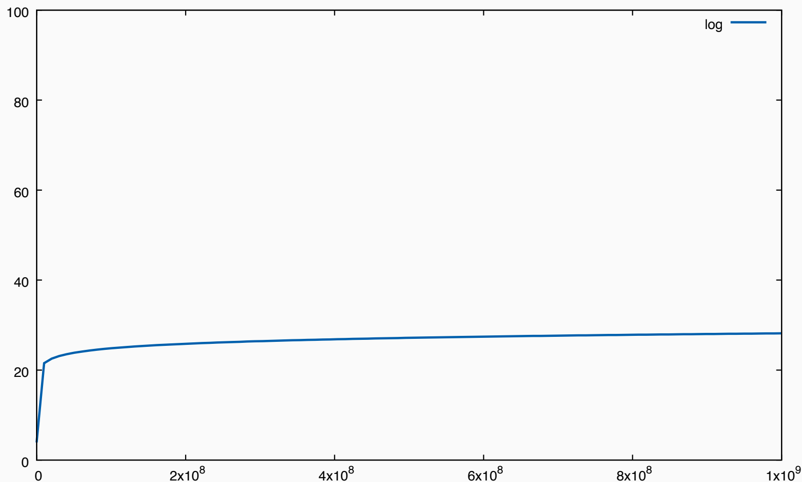
Complexité linéaire



Complexité logarithmique



Complexité logarithmique



Tri

Tri

- on peut donc chercher très vite dans un vecteur *si* il est trié
- question naturelle : comment trier un vecteur ?
- plusieurs algorithmes de tri existent
- aujourd'hui nous allons regarder un algorithme relativement simple
- sa complexité en temps n'est pas optimale

Tri par sélection

- idée : aller de gauche à droite et trouver l'élément le plus petit à droite de la position actuelle
- décompose le vecteur en
  - une partie déjà triée (à gauche de la position actuelle)
  - une partie pas encore triée (à droite de la position actuelle)

Tri par sélection

```
void tri(vector<int> &vec)
{
    for(int i = 0; i < vec.size(); ++i)
        for(int j = i; j < vec.size(); ++j)
            if(vec[j] < vec[i])
            {
                int temp = vec[i];
                vec[i] = vec[j];
                vec[j] = temp;
            }
}
```

Conclusion

- recherche dans un vecteur non trié a une complexité en temps linéaire en sa taille
- si le vecteur est trié, la recherche dichotomique prend temps logarithmique
- il est donc intéressant de trouver des algorithmes pour trier un vecteur

Questions